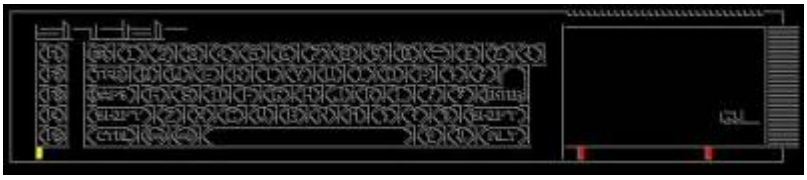

# SBASIC/SuperBASIC Reference Manual Online Documentation

***Release 4.0.2***

**Rich Mellor**

**Nov 07, 2025**

# CONTENTS


**1 Original Foreword** **3**
1.1 2015 Foreword . . . 4
1.2 Online Edition Foreword . . . 4

**2 Introduction** **5**
2.1 Contributing Authors . . . 7
2.2 Installing Toolkits . . . 7

**3 Credits** **9**
3.1 Other Notices . . . 10

**4 Structure of this Book** **11**
4.1 Syntax . . . 11
4.2 Description . . . 13
4.3 Examples . . . 13
4.4 Notes . . . 14
4.5 [Implementation] Notes . . . 14
4.6 Warning . . . 14
4.7 Cross-Reference . . . 14

**5 Writing Programs** **15**
5.1 Compiling SuperBASIC Programs . . . 15
5.2 Writing Programs to Run Under the Pointer Environment . . . 17
5.3 Multitasking Programs . . . 18

**6 Keywords Introduction** **19**

**7 Toolkits** **21**
7.1 Ähnlichkeiten . . . 21
7.2 ARRAY . . . 21
7.3 ATARI Emulators . . . 21
7.4 ATARIDOS . . . 22
7.5 ATARI_REXT . . . 22
7.6 Amiga QDOS - v3.20 . . . 24
7.7 BGI . . . 24
7.8 BIT . . . 25
7.9 BTool . . . 25
7.10 BeuleTools . . . 28
7.11 COMPICT . . . 29
7.12 CONCAT . . . 29

7.13 CONVERT . . . 29
7.14 CRYPTAGE . . . 30
7.15 DESPR . . . 30
7.16 DEV device . . . 30
7.17 DIY Toolkit . . . 30
7.18 Djtoolkit v1.16 . . . 35
7.19 Disk Interfaces . . . 36
7.20 ETAT . . . 36
7.21 Ecran Manager . . . 37
7.22 Environment Variables . . . 37
7.23 FACT . . . 37
7.24 FKEY . . . 37
7.25 FN . . . 37
7.26 FONTS . . . 38
7.27 FRACT . . . 38
7.28 Fast PLOT/DRAW Toolkit . . . 38
7.29 GETSTUFF . . . 38
7.30 Gold Card . . . 38
7.31 GPOINT . . . 40
7.32 HCO . . . 40
7.33 HOTKEY II . . . 40
7.34 Hard Disk Driver . . . 41
7.35 History Device . . . 41
7.36 Hyper . . . 41
7.37 Hyperbola . . . 42
7.38 KEYMAN . . . 42
7.39 KILL . . . 42
7.40 LWCUPC . . . 42
7.41 Level-2 Device Drivers . . . 42
7.42 MINMAX2 . . . 42
7.43 MULTI . . . 43
7.44 Math Package . . . 43
7.45 Minerva . . . 44
7.46 Minerva - Trace Toolkit . . . 44
7.47 Minerva Extensions Toolkit . . . 44
7.48 NDIM . . . 45
7.49 PAR/SER Interfaces . . . 45
7.50 PEX . . . 45
7.51 PICEXT . . . 45
7.52 PIE . . . 46
7.53 PRIO . . . 46
7.54 PTRRTP . . . 46
7.55 Path device . . . 46
7.56 Pointer Interface - v1.23 Onwards . . . 46
7.57 QL ROM . . . 47
7.58 QPC / QXL . . . 51
7.59 QSOUND . . . 52
7.60 QView Tiny Toolkit . . . 52
7.61 QVME - Level E-19 Drivers onwards . . . 53
7.62 QXL . . . 53
7.63 Qjump RAMPRT . . . 53

7.64 RES . . . 53
7.65 REV . . . 54
7.66 SDUMP_REXT . . . 54
7.67 SERMouse . . . 54
7.68 SMS . . . 54
7.69 SMSQ . . . 55
7.70 SMSQ/E . . . 56
7.71 ST/QL . . . 64
7.72 STAMP . . . 66
7.73 SWAP . . . 66
7.74 SYSBASE . . . 67
7.75 Shape Toolkit . . . 67
7.76 Super Gold Card . . . 67
7.77 SuperQBoard . . . 67
7.78 SuperWindow Toolkit . . . 67
7.79 THOR . . . 67
7.80 TRIM . . . 70
7.81 TRIPRODRO . . . 70
7.82 TRUFA . . . 70
7.83 TinyToolkit . . . 71
7.84 Toolfin . . . 73
7.85 Toolkit II . . . 73
7.86 Trump Card . . . 76
7.87 Turbo Toolkit . . . 77
7.88 UNJOB . . . 78
7.89 WIPE . . . 79
7.90 WM . . . 79
7.91 XKBD . . . 79

**8 Keywords A** **81**
8.1 ABS . . . 81
8.2 ABS_POSITION . . . 82
8.3 ACCEL_OFF . . . 83
8.4 ACCEL_ON . . . 83
8.5 ACCEL_SET . . . 83
8.6 ACCEL_STATE . . . 84
8.7 ACOPY . . . 84
8.8 ACOS . . . 84
8.9 ACOT . . . 85
8.10 ADATE . . . 86
8.11 ADDREG . . . 87
8.12 ADELETE . . . 87
8.13 ADIR . . . 88
8.14 AFORMAT . . . 88
8.15 AJOB . . . 88
8.16 ALARM . . . 89
8.17 ALCHP . . . 89
8.18 ALFM . . . 91
8.19 ALIAS . . . 91
8.20 ALINE . . . 93
8.21 ALLOCATION . . . 94

8.22 ALPHA_BLEND . . . 94
8.23 ALT . . . 95
8.24 ALTER . . . 95
8.25 ALTKEY . . . 96
8.26 AND . . . 97
8.27 APOINT . . . 98
8.28 APPEND . . . 98
8.29 AQCONVERT . . . 98
8.30 ARC . . . 98
8.31 ARC_R . . . 100
8.32 ARCOSH . . . 101
8.33 ARCOTH . . . 101
8.34 ARSINH . . . 101
8.35 ARTANH . . . 102
8.36 ASIN . . . 102
8.37 ASK . . . 103
8.38 ASTAT . . . 104
8.39 AT . . . 104
8.40 ATAN . . . 105
8.41 ATARI . . . 106
8.42 ATARI_EXT . . . 107
8.43 ATN . . . 107
8.44 ATN2 . . . 107
8.45 AUTO . . . 108
8.46 AUTO_DIS . . . 109
8.47 AUTO_TK2F1 . . . 110
8.48 AUTO_TK2F2 . . . 110
8.49 A_BLANK . . . 110
8.50 A_EMULATOR . . . 111
8.51 A_MACHINE . . . 111
8.52 A_OLDSCR . . . 112
8.53 A_PROCESSOR . . . 112
8.54 A_RDATE . . . 113
8.55 A_SDATE . . . 113
8.56 A_SPEED . . . 113

**9 Keywords B** **115**
9.1 BASIC . . . 115
9.2 BASICP . . . 115
9.3 BASIC_B . . . 116
9.4 BASIC_W . . . 116
9.5 BASIC_L . . . 116
9.6 BASIC_B% . . . 117
9.7 BASIC_W% . . . 117
9.8 BASIC_F . . . 117
9.9 BASIC_INDEX% . . . 117
9.10 BASIC_NAME$ . . . 118
9.11 BASIC_POINTER . . . 118
9.12 BASIC_TYPE% . . . 118
9.13 BAT . . . 119
9.14 BAT$ . . . 119

9.15 BAT_USE . . . 119
9.16 BAUD . . . 120
9.17 BAUDRATE . . . 123
9.18 BCLEAR . . . 124
9.19 BEEP . . . 124
9.20 BEEPING . . . 126
9.21 BELL . . . 127
9.22 Beule_EXT . . . 127
9.23 BGCOLOUR_QL . . . 127
9.24 BGCOLOUR_24 . . . 128
9.25 BGET . . . 128
9.26 BGIMAGE . . . 129
9.27 BICOP . . . 130
9.28 BIN . . . 130
9.29 BIN$ . . . 131
9.30 BINOM . . . 132
9.31 BIT% . . . 133
9.32 BLD . . . 134
9.33 BLOCK . . . 134
9.34 BLOOK . . . 135
9.35 BLS . . . 135
9.36 BMOVE . . . 136
9.37 BORDER . . . 136
9.38 BPEEK% . . . 137
9.39 BPEEK_W% . . . 138
9.40 BPEEK_L . . . 138
9.41 BPOKE . . . 138
9.42 BPOKE_W . . . 138
9.43 BPOKE_L . . . 138
9.44 BPUT . . . 139
9.45 BREAK_ON . . . 140
9.46 BREAK_OFF . . . 140
9.47 BREAK . . . 140
9.48 BREAK% . . . 141
9.49 BTool_EXT . . . 141
9.50 BTool_RMV . . . 142
9.51 BTRAP . . . 142
9.52 BUTTON% . . . 142
9.53 BVER$ . . . 143
9.54 BYTES_FREE . . . 143

**10 Keywords C** **145**
10.1 CACHE_ON . . . 145
10.2 CACHE_OFF . . . 145
10.3 CALL . . . 146
10.4 CAPS . . . 148
10.5 CARD_CREATE . . . 148
10.6 CARD_DIR$ . . . 150
10.7 CARD_INIT . . . 150
10.8 CARD_RENF . . . 150
10.9 CATNAP . . . 151

10.10 CBASE . . . 151
10.11 CCHR$ . . . 151
10.12 CDEC$ . . . 152
10.13 CD_ALLTIME . . . 152
10.14 CD_CLOSE . . . 153
10.15 CD_EJECT . . . 153
10.16 CD_FIRSTTRACK . . . 154
10.17 CD_HOUR . . . 154
10.18 CD_HSG2RED . . . 154
10.19 CD_INIT . . . 155
10.20 CD_ISCLOSED . . . 155
10.21 CD_ISINSERTED . . . 156
10.22 CD_ISPAUSED . . . 156
10.23 CD_ISPLAYING . . . 156
10.24 CD_LASTTRACK . . . 156
10.25 CD_LENGTH . . . 157
10.26 CD_MINUTE . . . 157
10.27 CD_PLAY . . . 157
10.28 CD_RED2HSG . . . 158
10.29 CD_RESUME . . . 158
10.30 CD_SECOND . . . 159
10.31 CD_STOP . . . 159
10.32 CD_TRACK . . . 160
10.33 CD_TRACKLENGTH . . . 160
10.34 CD_TRACKSTART . . . 160
10.35 CD_TRACKTIME . . . 160
10.36 CEIL . . . 161
10.37 CHANGE . . . 161
10.38 CHANID . . . 162
10.39 CHANNELS . . . 163
10.40 CHANNEL_ID . . . 163
10.41 CHAN_B% . . . 163
10.42 CHAN_W% . . . 163
10.43 CHAN_L% . . . 163
10.44 CHARGE . . . 164
10.45 CHAR_DEF . . . 165
10.46 CHAR_INC . . . 165
10.47 CHAR_USE . . . 166
10.48 CHBASE . . . 167
10.49 CHECK . . . 168
10.50 CHECK% . . . 168
10.51 CHECKF . . . 169
10.52 CHK_HEAP . . . 169
10.53 CHR$ . . . 170
10.54 CIRCLE . . . 170
10.55 CIRCLE_R . . . 171
10.56 CKEYOFF . . . 171
10.57 CKEYON . . . 172
10.58 CLCHP . . . 173
10.59 CLEAR . . . 173
10.60 CLEAR_HOT . . . 174

10.61 CLIP% . . . 174
10.62 CLIP$ . . . 176
10.63 CLOCK . . . 176
10.64 CLOSE . . . 177
10.65 CLOSE% . . . 179
10.66 CLRMDV . . . 179
10.67 CLS . . . 179
10.68 CLS_A . . . 180
10.69 CMD$ . . . 181
10.70 CODE . . . 182
10.71 CODEVEC . . . 182
10.72 COL . . . 183
10.73 COLOUR_NATIVE . . . 183
10.74 COLOUR_PAL . . . 184
10.75 COLOUR_QL . . . 185
10.76 COLOUR_24 . . . 186
10.77 COMMAND_LINE . . . 186
10.78 COMPILED . . . 187
10.79 COMPRESS . . . 187
10.80 CONCAT . . . 188
10.81 CONNECT . . . 188
10.82 CONTINUE . . . 189
10.83 ConvCASE$ . . . 189
10.84 CONVERT . . . 190
10.85 COPY . . . 190
10.86 COPY_B . . . 192
10.87 COPY_H . . . 193
10.88 COPY_L . . . 193
10.89 COPY_N . . . 194
10.90 COPY_O . . . 194
10.91 COPY_W . . . 194
10.92 COS . . . 195
10.93 COSH . . . 196
10.94 COT . . . 196
10.95 COTH . . . 198
10.96 CSIZE . . . 198
10.97 CTAB$ . . . 199
10.98 CUR . . . 199
10.99 CURDIS . . . 200
10.100 CURSEN . . . 200
10.101 CURSOR . . . 201
10.102 CURSOR% . . . 202
10.103 CURSOR_OFF . . . 203
10.104 CURSOR_ON . . . 203
10.105 CVF . . . 203
10.106 CVI% . . . 204
10.107 CVS$ . . . 204
10.108 CVL . . . 204

**11 Keywords D** **207**
11.1 DATA . . . 207

11.2 DATAD$ . . . 208
11.3 DATAREG . . . 209
11.4 DATASPACE . . . 209
11.5 DATA_AREA . . . 210
11.6 DATA_USE . . . 210
11.7 DATE . . . 212
11.8 DATE$ . . . 213
11.9 DAY$ . . . 215
11.10 DAY% . . . 215
11.11 DBL . . . 216
11.12 DDOWN . . . 216
11.13 DEALLOCATE . . . 217
11.14 DEBUG . . . 217
11.15 DEBUG_LEVEL . . . 217
11.16 DEFAULT . . . 218
11.17 DEFAULT% . . . 218
11.18 DEFAULT$ . . . 218
11.19 DEFAULT_DEVICE . . . 219
11.20 DEFAULT_SCR . . . 219
11.21 DEFine xxx . . . 220
11.22 DEFine FuNction . . . 220
11.23 DEFine PROCedure . . . 225
11.24 DEFINED . . . 227
11.25 DEG . . . 228
11.26 DELETE . . . 228
11.27 DEL_DEFB . . . 229
11.28 DESPR . . . 230
11.29 DESTD$ . . . 230
11.30 DEST_USE . . . 231
11.31 DEMO . . . 231
11.32 DET . . . 232
11.33 DEV_NAME . . . 233
11.34 DEVICE_SPACE . . . 234
11.35 DEVICE_STATUS . . . 235
11.36 DEVLIST . . . 236
11.37 DEVTYPE . . . 236
11.38 DEV_LIST . . . 236
11.39 DEV_NEXT . . . 237
11.40 DEV_USE . . . 238
11.41 DEV_USEN . . . 241
11.42 DEV_USE$ . . . 241
11.43 DIM . . . 242
11.44 DIMN . . . 251
11.45 DIR . . . 252
11.46 DISCARD . . . 255
11.47 DISP_BLANK . . . 255
11.48 DISP_INVERSE . . . 256
11.49 DISP_MODE . . . 256
11.50 DISP_RATE . . . 257
11.51 DISP_SIZE . . . 258
11.52 DISP_TYPE . . . 259

11.53 DISP_UPDATE . . . 259
11.54 DISPLAY_WIDTH . . . 260
11.55 DIV . . . 260
11.56 DIV . . . 261
11.57 DJ_OPEN . . . 262
11.58 DJ_OPEN_IN . . . 262
11.59 DJ_OPEN_NEW . . . 262
11.60 DJ_OPEN_OVER . . . 262
11.61 DJ_OPEN_DIR . . . 263
11.62 DJTK_VER$ . . . 264
11.63 DLINE . . . 264
11.64 DLIST . . . 265
11.65 DMEDIUM_DENSITY . . . 265
11.66 DMEDIUM_DRIVE$ . . . 266
11.67 DMEDIUM_FORMAT . . . 267
11.68 DMEDIUM_FREE . . . 267
11.69 DMEDIUM_NAME$ . . . 268
11.70 DMEDIUM_RDONLY . . . 269
11.71 DMEDIUM_REMOVE . . . 269
11.72 DMEDIUM_TOTAL . . . 269
11.73 DMEDIUM_TYPE . . . 270
11.74 DNEXT . . . 270
11.75 DO . . . 271
11.76 DOS_USE . . . 272
11.77 DOS_DRIVE . . . 273
11.78 DOS_DRIVE$ . . . 273
11.79 DOTLIN . . . 273
11.80 DRAW . . . 274
11.81 DRAW . . . 275
11.82 DROUND . . . 276
11.83 DUP . . . 276

**12 Keywords E** **277**
12.1 EASTER . . . 277
12.2 ED . . . 277
12.3 EDIT . . . 281
12.4 EDITF . . . 282
12.5 EDIT% . . . 283
12.6 EDIT$ . . . 283
12.7 EDLINE$ . . . 283
12.8 EL . . . 284
12.9 ELIS . . . 284
12.10 ELLIPSE . . . 285
12.11 ELLIPSE_R . . . 286
12.12 ELSE . . . 286
12.13 END . . . 287
12.14 END DEFine . . . 287
12.15 END FOR . . . 288
12.16 END IF . . . 289
12.17 END REPeat . . . 290
12.18 END SELect . . . 291

12.19 END WHEN . . . 291
12.20 END_CMD . . . 292
12.21 END_WHEN . . . 292
12.22 ENV_DEL . . . 292
12.23 ENV_LIST . . . 293
12.24 ENL . . . 293
12.25 EOF . . . 293
12.26 EOFW . . . 294
12.27 EPROM_LOAD . . . 295
12.28 EPS . . . 295
12.29 EQ$ . . . 296
12.30 ERLIN . . . 297
12.31 ERLIN% . . . 297
12.32 ERNUM . . . 297
12.33 ERNUM% . . . 299
12.34 ERR_XX . . . 300
12.35 ERRor . . . 301
12.36 ERT . . . 301
12.37 ESC . . . 303
12.38 ET . . . 303
12.39 ETAB$ . . . 303
12.40 ETAT . . . 304
12.41 EW . . . 305
12.42 EX . . . 308
12.43 EXCHG . . . 310
12.44 EXEC . . . 311
12.45 EXEC_W . . . 311
12.46 EXEP . . . 311
12.47 EXIT . . . 314
12.48 EXP . . . 315
12.49 EXPAND . . . 316
12.50 EXPLODE . . . 316
12.51 EXTRAS . . . 317
12.52 EXTRAS_W . . . 317

**13 Keywords F** **319**
13.1 FACT . . . 319
13.2 FALSE% . . . 319
13.3 FASTEXPAND . . . 320
13.4 FAT_DRIVE . . . 320
13.5 FAT_DRIVE$ . . . 321
13.6 FAT_USE . . . 321
13.7 FAT_WP . . . 322
13.8 FBKDT . . . 322
13.9 FDAT . . . 325
13.10 FDEC$ . . . 325
13.11 FETCH_BYTES . . . 325
13.12 FEXP$ . . . 326
13.13 FET . . . 326
13.14 FEW . . . 327
13.15 FEX . . . 327

13.16 FEX_M . . . 327
13.17 FF . . . 328
13.18 FGET% . . . 328
13.19 FGET$ . . . 328
13.20 FGETB . . . 329
13.21 FGETL . . . 329
13.22 FGETF . . . 330
13.23 FGETH$ . . . 330
13.24 FILE_BACKUP . . . 332
13.25 FILE_DAT . . . 332
13.26 FILE_DATASPACE . . . 332
13.27 FILE_LEN . . . 333
13.28 FILE_LENGTH . . . 334
13.29 FILE_OPEN . . . 334
13.30 FILE_POS . . . 335
13.31 FILE_POSITION . . . 335
13.32 FILE_PTRA . . . 336
13.33 FILE_PTRR . . . 336
13.34 FILE_TYPE . . . 337
13.35 FILE_UPDATE . . . 338
13.36 FILL . . . 338
13.37 FILL$ . . . 340
13.38 FILLMEM_B . . . 341
13.39 FILLMEM_W . . . 341
13.40 FILLMEM_L . . . 342
13.41 FIND . . . 342
13.42 FLASH . . . 343
13.43 FLEN . . . 344
13.44 FLIS . . . 344
13.45 FLP_DENSITY . . . 345
13.46 FLP_DRIVE . . . 346
13.47 FLP_DRIVE$ . . . 347
13.48 FLP_EXT . . . 347
13.49 FLP_JIGGLE . . . 347
13.50 FLP_SEC . . . 348
13.51 FLP_START . . . 349
13.52 FLP_STEP . . . 349
13.53 FLP_TRACK . . . 350
13.54 FLP_USE . . . 350
13.55 FLUSH . . . 351
13.56 FLUSH_CHANNEL . . . 351
13.57 FMAKE_DIR . . . 352
13.58 FNAME$ . . . 352
13.59 FOPEN . . . 353
13.60 FOP_DIR . . . 353
13.61 FOP_IN . . . 354
13.62 FOP_NEW . . . 354
13.63 FOP_OVER . . . 354
13.64 FOR . . . 355
13.65 FORCE_TYPE . . . 359
13.66 FORMAT . . . 360

13.67 FPOS . . . 365
13.68 FPOS_A . . . 366
13.69 FPOS_R . . . 366
13.70 FPUT$ . . . 366
13.71 FPUT% . . . 366
13.72 FPUTB . . . 367
13.73 FPUTF . . . 367
13.74 FPUTL . . . 367
13.75 FRACT . . . 368
13.76 FREAD . . . 368
13.77 FREAD$ . . . 369
13.78 FREE . . . 369
13.79 FREE_FAST . . . 370
13.80 FREE_FMEM . . . 370
13.81 FREE_MEM . . . 371
13.82 FREEZE . . . 371
13.83 FREEZE% . . . 371
13.84 FSERVE . . . 372
13.85 FSETH$ . . . 374
13.86 FTEST . . . 374
13.87 FTYP . . . 375
13.88 FuNction . . . 376
13.89 FUPDT . . . 376
13.90 FVERS . . . 378
13.91 FWRITE . . . 379
13.92 FWRITE$ . . . 379
13.93 FXTRA . . . 380

**14 Keywords G** **381**
14.1 GCD . . . 381
14.2 GER_MSG . . . 381
14.3 GER_TRA . . . 381
14.4 GET . . . 382
14.5 GET_BYTE$ . . . 383
14.6 GET_BYTE . . . 384
14.7 GET_FLOAT . . . 385
14.8 GET_LONG . . . 385
14.9 GET_STRING . . . 385
14.10 GET_STUFF$ . . . 386
14.11 GetHEAD . . . 386
14.12 GET_WORD . . . 387
14.13 GETXY . . . 387
14.14 GO SUB . . . 388
14.15 GO TO . . . 389
14.16 GPOINT . . . 390
14.17 GRAB . . . 390
14.18 GREGOR . . . 390
14.19 GT$ . . . 392

**15 Keywords H** **393**
15.1 HEADR . . . 393
15.2 HEADS . . . 393

15.3 HEX . . . 393
15.4 HEX$ . . . 394
15.5 HGET . . . 395
15.6 HIS_SET . . . 395
15.7 HIS_SIZE . . . 396
15.8 HIS_UNSET . . . 397
15.9 HIS_USE . . . 397
15.10 HIS_USE$ . . . 398
15.11 HOT . . . 398
15.12 HOT_CHP . . . 399
15.13 HOT_CHP1 . . . 400
15.14 HOT_CMD . . . 401
15.15 HOT_DO . . . 401
15.16 HOT_GETSTUFF$ . . . 402
15.17 HOT_GO . . . 402
15.18 HOT_KEY . . . 403
15.19 HOT_LIST . . . 403
15.20 HOT_LOAD . . . 403
15.21 HOT_LOAD1 . . . 404
15.22 HOT_NAME$ . . . 404
15.23 HOT_OFF . . . 405
15.24 HOT_PICK . . . 405
15.25 HOT_REMV . . . 406
15.26 HOT_RES . . . 406
15.27 HOT_RES1 . . . 407
15.28 HOT_SET . . . 407
15.29 HOT_STOP . . . 407
15.30 HOT_STUFF . . . 408
15.31 HOT_THING . . . 409
15.32 HOT_THING1 . . . 409
15.33 HOT_TYPE . . . 410
15.34 HOT_WAKE . . . 410
15.35 HPUT . . . 411

**16 Keywords I** **413**
16.1 I2C_IO . . . 413
16.2 IDEC$ . . . 415
16.3 IF . . . 415
16.4 IFORMAT . . . 418
16.5 INARRAY% . . . 419
16.6 INF . . . 420
16.7 INK . . . 420
16.8 INKEY$ . . . 423
16.9 INPUT . . . 424
16.10 INPUT$ . . . 430
16.11 INSTR . . . 430
16.12 INSTR_CASE . . . 430
16.13 INT . . . 431
16.14 INTMAX . . . 432
16.15 INVERSE . . . 432
16.16 INVXY . . . 433

16.17 IO_PEND% . . . 433
16.18 IO_PRIORITY . . . 433
16.19 IO_TRAP . . . 433
16.20 IQCONVERT . . . 434
16.21 IS_BASIC . . . 435
16.22 IS_PEON . . . 435
16.23 IS_PTRAP . . . 436

**17 Keywords J** **439**
17.1 JBASE . . . 439
17.2 JobCBS . . . 441
17.3 JOBID . . . 441
17.4 JOBS . . . 441
17.5 JOB$ . . . 442
17.6 JOB_NAME . . . 442

**18 Keywords K** **445**
18.1 KBD_RESET . . . 445
18.2 KBD_TABLE . . . 445
18.3 KBD_USE . . . 446
18.4 KBYTES_FREE . . . 446
18.5 KEY . . . 446
18.6 KEYROW . . . 448
18.7 KEYW . . . 452
18.8 KEY_ADD . . . 452
18.9 KEY_RMV . . . 454
18.10 KILL . . . 454
18.11 KILLN . . . 454
18.12 KILLSOUND . . . 455
18.13 KILL_A . . . 455
18.14 KJOB . . . 455
18.15 KJOBS . . . 456

**19 Keywords L** **457**
19.1 LANG_USE . . . 457
19.2 LANGUAGE . . . 457
19.3 LANGUAGE$ . . . 458
19.4 LAR . . . 458
19.5 LBYTES . . . 458
19.6 LCM . . . 459
19.7 LDRAW . . . 460
19.8 LEFT . . . 462
19.9 LEN . . . 462
19.10 LET . . . 463
19.11 LEVEL2 . . . 464
19.12 LGET . . . 465
19.13 LINE . . . 465
19.14 LINE_R . . . 466
19.15 LINKUP . . . 467
19.16 LINT2 . . . 467
19.17 LIST . . . 467
19.18 LIST_TASKS . . . 468

19.19 LMAR . . . 468
19.20 LN . . . 469
19.21 LOAD . . . 469
19.22 LOADPIC . . . 472
19.23 LOCal . . . 472
19.24 LOCK . . . 474
19.25 LOG2 . . . 474
19.26 LOG10 . . . 475
19.27 LOOKUP% . . . 475
19.28 LOWER$ . . . 476
19.29 LPOLL . . . 476
19.30 LPR_USE . . . 477
19.31 LPUT . . . 477
19.32 LRESFAST . . . 477
19.33 LRESPR . . . 478
19.34 LRUN . . . 479
19.35 LSCHD . . . 479
19.36 LWC$ . . . 479

**20 Keywords M** **481**
20.1 MACHINE . . . 481
20.2 MAKE_DIR . . . 484
20.3 MATADD . . . 486
20.4 MATCOUNT . . . 486
20.5 MATCOUNT1 . . . 487
20.6 MATEQU . . . 487
20.7 MATDEV . . . 488
20.8 MATIDN . . . 488
20.9 MATINPUT . . . 489
20.10 MATINV . . . 489
20.11 MATMAX . . . 491
20.12 MATMEAN . . . 491
20.13 MATMIN . . . 491
20.14 MATMULT . . . 492
20.15 MATPLOT . . . 494
20.16 MATPLOT_R . . . 495
20.17 MATPROD . . . 495
20.18 MATREAD . . . 496
20.19 MATRND . . . 497
20.20 MATSEQ . . . 497
20.21 MATSUB . . . 498
20.22 MATSUM . . . 498
20.23 MATTRN . . . 499
20.24 MAX . . . 500
20.25 MAX_CON . . . 500
20.26 MAX_DEVS . . . 501
20.27 MAXIMUM . . . 502
20.28 MAXIMUM% . . . 502
20.29 MB . . . 503
20.30 MD . . . 504
20.31 MERGE . . . 504

20.32 MIDINET . . . 505
20.33 MIN . . . 506
20.34 MINIMUM . . . 506
20.35 MINIMUM% . . . 507
20.36 MISTake . . . 508
20.37 MKF$ . . . 508
20.38 MKI$ . . . 508
20.39 MKL$ . . . 509
20.40 MKS$ . . . 509
20.41 MNET . . . 509
20.42 MNET% . . . 510
20.43 MNET_OFF . . . 510
20.44 MNET_ON . . . 510
20.45 MNET_S% . . . 510
20.46 MNET_USE . . . 511
20.47 MOD . . . 511
20.48 MOD . . . 512
20.49 MODE . . . 512
20.50 MONTH% . . . 518
20.51 MORE . . . 518
20.52 MOUSE_SPEED . . . 519
20.53 MOUSE_STUFF . . . 519
20.54 MOVE . . . 520
20.55 MOVE_MEM . . . 521
20.56 MOVE_POSITION . . . 521
20.57 MRUN . . . 522
20.58 MSEARCH . . . 522
20.59 MT . . . 522
20.60 MTRAP . . . 523

**21 Keywords N** **525**
21.1 NDIM . . . 525
21.2 NDIM% . . . 525
21.3 NET . . . 526
21.4 NETBEEP . . . 526
21.5 NETPOLL . . . 527
21.6 NETRATE . . . 527
21.7 NETREAD . . . 529
21.8 NETSEND . . . 529
21.9 NETVAR% . . . 530
21.10 NET_ID . . . 530
21.11 NEW . . . 531
21.12 NEWCHAN% . . . 531
21.13 NEW_NAME . . . 532
21.14 NEXT . . . 533
21.15 NFS_USE . . . 533
21.16 NIX . . . 534
21.17 NO_CLOCK . . . 535
21.18 NOCAPS . . . 535
21.19 NOKEY . . . 535
21.20 NORM . . . 535

21.21 NOR_MSG . . . 536
21.22 NOR_TRA . . . 536
21.23 NOT . . . 536
21.24 NRM . . . 538
21.25 NXJOB . . . 538

**22 Keywords O** **541**
22.1 ODD . . . 541
22.2 OFF . . . 541
22.3 OJOB . . . 541
22.4 ON . . . 542
22.5 ON…GO TO . . . 542
22.6 ON…GO SUB . . . 542
22.7 OPEN . . . 543
22.8 OPEN_DIR . . . 546
22.9 OPEN_IN . . . 550
22.10 OPEN_NEW . . . 551
22.11 OPEN_OVER . . . 552
22.12 OR . . . 552
22.13 OUTL . . . 553
22.14 OUTLN . . . 553
22.15 OVER . . . 557

**23 Keywords P** **559**
23.1 PAGDIS . . . 559
23.2 PAGLEN . . . 559
23.3 PAGLIN . . . 560
23.4 PAINT . . . 560
23.5 PALETTE_QL . . . 561
23.6 PALETTE_8 . . . 562
23.7 PAN . . . 562
23.8 PAPER . . . 564
23.9 PARHASH . . . 564
23.10 PARNAM$ . . . 566
23.11 PARNAME$ . . . 566
23.12 PARSEPA . . . 567
23.13 PARSTR$ . . . 567
23.14 PARTYP . . . 568
23.15 PARTYPE . . . 570
23.16 PARUSE . . . 570
23.17 PAR_ABORT . . . 571
23.18 PAR_BUFF . . . 571
23.19 PAR_CLEAR . . . 572
23.20 PAR_DEFAULTPRINTER$ . . . 572
23.21 PAR_GETFILTER . . . 572
23.22 PAR_GETPRINTER$ . . . 573
23.23 PAR_PRINTERCOUNT . . . 573
23.24 PAR_PRINTERNAME$ . . . 573
23.25 PAR_PULSE . . . 573
23.26 PAR_SETFILTER . . . 573
23.27 PAR_SETPRINTER . . . 574
23.28 PAR_USE . . . 574

23.29 PAUSE . . . 574
23.30 PE_BGOFF . . . 575
23.31 PE_BGON . . . 575
23.32 PEEK . . . 576
23.33 PEEK_FLOAT . . . 576
23.34 PEEK_STRING . . . 577
23.35 PEEK_W . . . 578
23.36 PEEK_L . . . 578
23.37 PEEKS . . . 579
23.38 PEEKS_W . . . 579
23.39 PEEKS_L . . . 579
23.40 PEEK$ . . . 580
23.41 PEEK_F . . . 581
23.42 PEND . . . 581
23.43 PENDOWN . . . 583
23.44 PENUP . . . 583
23.45 PEOFF . . . 583
23.46 PEON . . . 583
23.47 PEX$ . . . 584
23.48 PEX_INI . . . 585
23.49 PEX_SAVE . . . 585
23.50 PEX_XTD . . . 585
23.51 PHONEM . . . 586
23.52 PI . . . 586
23.53 PICK$ . . . 587
23.54 PICK% . . . 588
23.55 PIE_EX_OFF . . . 589
23.56 PIE_EX_ON . . . 589
23.57 PIE_OFF . . . 589
23.58 PIE_ON . . . 590
23.59 PIF$ . . . 590
23.60 PINF$ . . . 590
23.61 PIXEL% . . . 590
23.62 PJOB . . . 591
23.63 PLAY . . . 591
23.64 PLOT . . . 592
23.65 PLOT . . . 592
23.66 POINT . . . 593
23.67 POINT_R . . . 594
23.68 POKE . . . 594
23.69 POKE_FLOAT . . . 594
23.70 POKE_STRING . . . 594
23.71 POKE_W . . . 595
23.72 POKE_L . . . 595
23.73 POKES . . . 599
23.74 POKES_W . . . 599
23.75 POKES_L . . . 600
23.76 POKE$ . . . 600
23.77 POKE_F . . . 601
23.78 PRINT . . . 601
23.79 PRINT_USING . . . 603

23.80 PRIO . . . 606
23.81 PRIORITISE . . . 607
23.82 PRO . . . 607
23.83 PROCESSOR . . . 607
23.84 PROCedure . . . 608
23.85 PROGD$ . . . 609
23.86 PROG_USE . . . 609
23.87 PROT_DATE . . . 609
23.88 PROT_MEM . . . 610
23.89 PROUND . . . 611
23.90 PRT_ABORT . . . 612
23.91 PRT_ABT . . . 612
23.92 PRT_BUFF . . . 612
23.93 PRT_CLEAR . . . 612
23.94 PRT_USE . . . 613
23.95 PRT_USE . . . 613
23.96 PRT_USE$ . . . 614
23.97 PTH_ADD . . . 615
23.98 PTH_LIST . . . 617
23.99 PTH_RMV . . . 617
23.100 PTH_USE . . . 618
23.101 PTH_USE$ . . . 618
23.102 PTH$ . . . 619
23.103 PTR_FN% . . . 619
23.104 PTR_INC . . . 620
23.105 PTR_KEY . . . 620
23.106 PTR_LIMITS . . . 621
23.107 PTR_MAX . . . 621
23.108 PTR_OFF . . . 622
23.109 PTR_ON . . . 622
23.110 PTR_POS . . . 622
23.111 PTR_X . . . 623
23.112 PTR_Y . . . 623
23.113 PURGE . . . 623
23.114 PUT . . . 623
23.115 PUT_BYTE . . . 624
23.116 PUT_FLOAT . . . 625
23.117 PUT_LONG . . . 625
23.118 PUT_STRING . . . 625
23.119 PUT_WORD . . . 626
23.120 PXOFF . . . 626
23.121 PXON . . . 626
23.122 PX1ST . . . 627
23.123 P_ENV . . . 627

**24 Keywords Q** **629**
24.1 QACONVERT . . . 629
24.2 QCOPY . . . 629
24.3 QCOUNT% . . . 630
24.4 QDOS$ . . . 630
24.5 QFLIM . . . 631

24.6 QICONVERT . . . 632
24.7 QLINK . . . 633
24.8 QLOAD . . . 633
24.9 QLRUN . . . 633
24.10 QL_PEX . . . 634
24.11 QMERGE . . . 634
24.12 QMRUN . . . 634
24.13 QPC_CMDLINE$ . . . 634
24.14 QPC_EXEC . . . 635
24.15 QPC_EXIT . . . 635
24.16 QPC_HOSTOS . . . 635
24.17 QPC_MAXIMIZE . . . 635
24.18 QPC_MINIMIZE . . . 636
24.19 QPC_MSPEED . . . 636
24.20 QPC_NETNAME$ . . . 636
24.21 QPC_QLSCREMU . . . 636
24.22 QPC_RESTORE . . . 637
24.23 QPC_SYNCSCRAP . . . 637
24.24 QPC_VER$ . . . 637
24.25 QPC_WINDOWSIZE . . . 637
24.26 QPC_WINDOWTITLE . . . 638
24.27 QPTR . . . 638
24.28 QRAM$ . . . 638
24.29 QSAVE . . . 639
24.30 QSAVE_O . . . 640
24.31 QSIZE% . . . 640
24.32 QSPACE% . . . 641
24.33 QTRAP . . . 641
24.34 QuATARI . . . 642
24.35 QUB_DRIVE . . . 642
24.36 QUB_DRIVE$ . . . 642
24.37 QUB_USE . . . 643
24.38 QUB_WP . . . 644
24.39 QUEUE% . . . 644
24.40 QUIT . . . 644

**25 Keywords R** **647**
25.1 RAD . . . 647
25.2 RAE . . . 647
25.3 RAFE . . . 648
25.4 RAMTOP . . . 648
25.5 RAM_USE . . . 648
25.6 RAND . . . 648
25.7 RANDOMISE . . . 649
25.8 READ . . . 650
25.9 READ_HEADER . . . 651
25.10 RECHP . . . 653
25.11 RECOL . . . 654
25.12 REFRESH . . . 655
25.13 RELEASE . . . 655
25.14 RELEASE . . . 656

25.15 RELEASE_HEAP . . . 656
25.16 RELEASE_TASK . . . 656
25.17 RELJOB . . . 656
25.18 RELOAD . . . 657
25.19 REL_JOB . . . 657
25.20 REMAINDER . . . 658
25.21 REMark . . . 658
25.22 REMOVE . . . 658
25.23 REMOVE_TASK . . . 659
25.24 RENAME . . . 659
25.25 RENUM . . . 660
25.26 REPeat . . . 663
25.27 REPLACE . . . 665
25.28 REPLY . . . 667
25.29 REPORT . . . 667
25.30 RESAVE . . . 668
25.31 RESERVE . . . 668
25.32 RESERVE_HEAP . . . 669
25.33 RESET . . . 669
25.34 RESFAST . . . 670
25.35 RESPR . . . 670
25.36 RESTORE . . . 671
25.37 RES_SIZE . . . 671
25.38 RES_128 . . . 672
25.39 RETRY . . . 673
25.40 RETurn . . . 673
25.41 REV$ . . . 674
25.42 RJOB . . . 674
25.43 RMAR . . . 675
25.44 RMODE . . . 675
25.45 RND . . . 676
25.46 ROM . . . 677
25.47 ROM_EXT . . . 677
25.48 ROM_LOAD . . . 678
25.49 ROMs . . . 678
25.50 RTP_R . . . 678
25.51 RTP_T . . . 679
25.52 RUN . . . 679

**26 Keywords S** **681**
26.1 SAR . . . 681
26.2 SARO . . . 681
26.3 SAUTO . . . 682
26.4 SAVE . . . 682
26.5 SAVE_O . . . 683
26.6 SAVEPIC . . . 684
26.7 SB_THING . . . 684
26.8 SBASIC . . . 684
26.9 SBYTES . . . 686
26.10 SBYTES_O . . . 688
26.11 SCALE . . . 688

26.12 SCLR . . . 691
26.13 SCRBASE . . . 692
26.14 SCREEN . . . 693
26.15 SCREEN_BASE . . . 693
26.16 SCREEN_MODE . . . 694
26.17 SCRINC . . . 694
26.18 SCROLL . . . 695
26.19 SCROF . . . 696
26.20 SCRON . . . 697
26.21 SCR2DIS . . . 697
26.22 SCR2EN . . . 698
26.23 SCR_BASE . . . 698
26.24 SCR_LLEN . . . 698
26.25 SCR_REFRESH . . . 699
26.26 SCR_SAVE . . . 699
26.27 SCR_SIZE . . . 700
26.28 SCR_STORE . . . 700
26.29 SCR_XLIM . . . 701
26.30 SCR_YLIM . . . 701
26.31 SDATE . . . 702
26.32 SDP_DEV . . . 703
26.33 SDP_KEY . . . 703
26.34 SDP_SET . . . 704
26.35 SDUMP . . . 711
26.36 SEARCH . . . 712
26.37 SEARCH . . . 713
26.38 SEARCH_C . . . 714
26.39 SEARCH_I . . . 714
26.40 SEARCH_MEM . . . 715
26.41 SELect . . . 715
26.42 SELect ON . . . 715
26.43 SEND_EVENT . . . 719
26.44 SERMAWS . . . 719
26.45 SERMCUR . . . 720
26.46 SERMOFF . . . 720
26.47 SERMON . . . 720
26.48 SERMPTR . . . 721
26.49 SERMRESET . . . 721
26.50 SERMSPEED . . . 721
26.51 SERMWAIT . . . 722
26.52 SERNET . . . 722
26.53 SER_ABORT . . . 723
26.54 SER_BUFF . . . 723
26.55 SER_CDEOF . . . 724
26.56 SER_CLEAR . . . 724
26.57 SER_FLOW . . . 724
26.58 SER_GETPORT$ . . . 725
26.59 SER_PAUSE . . . 725
26.60 SER_ROOM . . . 726
26.61 SER_SETPORT . . . 726
26.62 SER_USE . . . 726

26.63 SET . . . 727
26.64 SET . . . 727
26.65 SetHEAD . . . 728
26.66 SET_HEADER . . . 729
26.67 SET_CLOCK . . . 729
26.68 SET_FBKDT . . . 729
26.69 SET_FUPDT . . . 730
26.70 SET_FVERS . . . 730
26.71 SET_GREEN . . . 731
26.72 SET_RED . . . 731
26.73 SET_LANGUAGE . . . 732
26.74 SET_XINC . . . 733
26.75 SET_YINC . . . 733
26.76 SEXEC . . . 733
26.77 SEXEC_O . . . 735
26.78 SGN . . . 735
26.79 SGN% . . . 735
26.80 SHOOT . . . 735
26.81 SI . . . 736
26.82 SIGN . . . 736
26.83 SIN . . . 736
26.84 SINH . . . 737
26.85 SINT . . . 737
26.86 SIZE . . . 738
26.87 SJOB . . . 740
26.88 SLOAD . . . 741
26.89 SLUG . . . 742
26.90 SMOVE . . . 742
26.91 SND_EXT . . . 743
26.92 SNET . . . 743
26.93 SNET% . . . 744
26.94 SNET_ROPEN . . . 744
26.95 SNET_S% . . . 744
26.96 SNET_USE . . . 744
26.97 SORT . . . 745
26.98 SOUNDEX . . . 746
26.99 SPJOB . . . 747
26.100 SPL . . . 748
26.101 SPLF . . . 748
26.102 SPL_USE . . . 749
26.103 SP_JOB . . . 749
26.104 SQR . . . 750
26.105 SQRT . . . 750
26.106 SSAVE . . . 750
26.107 SSHOW . . . 751
26.108 SSTAT . . . 751
26.109 SSTEP . . . 751
26.110 STAMP . . . 752
26.111 STAT . . . 752
26.112 STEP . . . 753
26.113 STOP . . . 753

26.114 STRIP . . . 754
26.115 SUB . . . 754
26.116 SOUNDFILE . . . 755
26.117 SOUNDFILE2 . . . 755
26.118 SOUNDFILE3 . . . 755
26.119 SUSJOB . . . 756
26.120 SWAP . . . 756
26.121 SXTRAS . . . 757
26.122 SYNCH% . . . 757
26.123 SYSBASE . . . 757
26.124 SYS_BASE . . . 758
26.125 SYS_VARS . . . 758
26.126 S_FONT . . . 759
26.127 S_LOAD . . . 759
26.128 S_SAVE . . . 759
26.129 S_SHOW . . . 760
26.130 SYSTEM_VARIABLES . . . 760

**27 Keywords T** **761**
27.1 TAN . . . 761
27.2 TANH . . . 762
27.3 TCA . . . 762
27.4 TCONNECT . . . 762
27.5 TEE . . . 764
27.6 THEN . . . 764
27.7 THING . . . 764
27.8 TH_FIX . . . 765
27.9 TH_VER$ . . . 765
27.10 TINY_EXT . . . 766
27.11 TINY_RMV . . . 766
27.12 TK2_EXT . . . 766
27.13 TK_VER$ . . . 767
27.14 TNC . . . 767
27.15 TO . . . 767
27.16 TOP_WINDOW . . . 768
27.17 TPFree . . . 768
27.18 TRA . . . 768
27.19 TRIM$ . . . 776
27.20 TRINT . . . 776
27.21 TROFF . . . 777
27.22 TRON . . . 777
27.23 TRUE% . . . 777
27.24 TRUNCATE . . . 778
27.25 TTALL . . . 778
27.26 TTEDELETE . . . 778
27.27 TTEFP . . . 779
27.28 TTEOPEN . . . 779
27.29 TTET3 . . . 780
27.30 TTEX . . . 781
27.31 TTEX_W . . . 782
27.32 TTFINDM . . . 782

27.33 TTINC . . . 783
27.34 TTME% . . . 783
27.35 TTMODE% . . . 783
27.36 TTPEEK$ . . . 784
27.37 TTPOKEM . . . 784
27.38 TTPOKE$ . . . 784
27.39 TTREL . . . 784
27.40 TTRENAME . . . 785
27.41 TTSUS . . . 785
27.42 TTV . . . 785
27.43 TT$ . . . 786
27.44 TURBO_diags . . . 787
27.45 TURBO_F . . . 787
27.46 TURBO_locstr . . . 788
27.47 TURBO_model . . . 788
27.48 TURBO_objdat . . . 789
27.49 TURBO_objfil . . . 789
27.50 TURBO_optim . . . 790
27.51 TURBO_P . . . 790
27.52 TURBO_repfil . . . 791
27.53 TURBO_struct . . . 791
27.54 TURBO_taskn . . . 792
27.55 TURBO_window . . . 792
27.56 TURN . . . 793
27.57 TURNTO . . . 793
27.58 TXTRAS . . . 794
27.59 TYPE . . . 794
27.60 TYPE_IN . . . 795
27.61 T_COUNT . . . 796
27.62 T_OFF . . . 796
27.63 T_ON . . . 796
27.64 T_RESTART . . . 797
27.65 T_START . . . 797
27.66 T_STOP . . . 797

**28 Keywords U** **799**
28.1 UINT . . . 799
28.2 UNDER . . . 799
28.3 UNJOB . . . 800
28.4 UNL . . . 800
28.5 UNLOAD . . . 801
28.6 UNLOCK . . . 802
28.7 UNSET . . . 802
28.8 UPC$ . . . 803
28.9 UPPER$ . . . 803
28.10 UPUT . . . 803
28.11 USE . . . 804
28.12 USE_FONT . . . 804

**29 Keywords V** **807**
29.1 VA . . . 807
29.2 VAR . . . 807

29.3 VER$ . . . 807
29.4 VFR . . . 810
29.5 VG_HOCH . . . 811
29.6 VG_LOAD . . . 811
29.7 VG_PARA . . . 811
29.8 VG_PRINT . . . 813
29.9 VG_RESO . . . 813
29.10 VG_WIND . . . 814
29.11 VIEW . . . 814
29.12 VOCAB . . . 815

**30 Keywords W** **817**
30.1 WAIT_EVENT . . . 817
30.2 WBASE . . . 817
30.3 WCOPY . . . 818
30.4 WCOPY_F . . . 821
30.5 WCOPY_O . . . 822
30.6 WDEL . . . 822
30.7 WDEL_F . . . 823
30.8 WDIR . . . 823
30.9 WEEKDAY% . . . 823
30.10 WGET . . . 824
30.11 WHEN condition . . . 824
30.12 WHEN ERRor . . . 826
30.13 WHERE_FONTS . . . 828
30.14 WIDTH . . . 829
30.15 WINDOW . . . 830
30.16 WINF$ . . . 831
30.17 WIN2 . . . 831
30.18 WIN_BASE . . . 832
30.19 WIN_CHECK . . . 832
30.20 WIN_DRIVE . . . 833
30.21 WIN_DRIVE$ . . . 834
30.22 WIN_FORMAT . . . 835
30.23 WIN_REMV . . . 836
30.24 WIN_SAFE . . . 837
30.25 WIN_SLUG . . . 837
30.26 WIN_START . . . 837
30.27 WIN_STOP . . . 838
30.28 WIN_USE . . . 838
30.29 WIN_WP . . . 839
30.30 WIPE . . . 839
30.31 WLD . . . 840
30.32 WM . . . 841
30.33 WM_BLOCK . . . 841
30.34 WM_BORDER . . . 842
30.35 WM_INK . . . 842
30.36 WM_MOVEMODE . . . 844
30.37 WM_PAPER . . . 845
30.38 WM_STRIP . . . 845
30.39 WMAN$ . . . 846

30.40 WMON . . . 847
30.41 WMOV . . . 848
30.42 WPUT . . . 848
30.43 WREN . . . 849
30.44 WSET . . . 849
30.45 WSET_DEF . . . 850
30.46 WSTAT . . . 850
30.47 WTV . . . 851
30.48 W_CRUNCH . . . 851
30.49 W_SHOW . . . 853
30.50 W_STORE . . . 853
30.51 W_SWAP . . . 854
30.52 W_SWOP . . . 854

**31 Keywords X** **855**
31.1 XCHANGE . . . 855
31.2 XDRAW . . . 855
31.3 XLIM . . . 856
31.4 XOR . . . 856
31.5 X_PTR% . . . 857

**32 Keywords Y** **859**
32.1 YEAR% . . . 859
32.2 YLIM . . . 859
32.3 Y_PTR% . . . 860

**33 Keywords Z** **861**
33.1 ZAP . . . 861

**34 Keywords Other** **863**
34.1 _DEF% . . . 863
34.2 _DEF$ . . . 863
34.3 _NAME$ . . . 864

**35 Appendices Introduction** **865**

**36 A1. Minerva** **867**
36.1 A1.1 INTRODUCTION . . . 867
36.2 A1.2 Windows and Closing Windows . . . 867
36.3 A1.3 Dual Screen Mode . . . 868
36.4 A1.4 Border . . . 868
36.5 A1.5 Empty Brackets . . . 869
36.6 A1.7 MultiBASICs . . . 869
36.7 A1.8 Strings . . . 870

**37 A2 SMSQ/E** **871**
37.1 A2.1 Introduction . . . 871
37.2 A2.2 The EOF Function . . . 872
37.3 A2.3 Empty Brackets . . . 872
37.4 A2.4 Multiple Sbasics . . . 873
37.5 A2.5 Improved Interpreter . . . 873
37.6 A2.6 Numbers in Programs . . . 873
37.7 A2.7 Inbuilt Pointer Environment . . . 873

37.8 A2.8 Undefined Variables . . . 873
37.9 A2.9 Extended Display . . . 874
37.10 A2.10 Problems . . . 874

**38 A3 Emulators 877**
38.1 A3.1 Introduction . . . 877
38.2 A3.2 Apple Macintosh . . . 878
38.3 A3.3 IBM Compatible PCs . . . 878
38.4 A3.4 Atari Computers . . . 881
38.5 A3.4.1 The ST/QL Emulator . . . 882
38.6 A3.4.2 SMSQ/E . . . 884
38.7 A3.4.3 SMS2 . . . 885
38.8 A3.5 Commodore Amigas . . . 885
38.9 A3.6 Unix Systems . . . 887

**39 A4 Thor Computers 889**
39.1 A4.1 Introduction . . . 889
39.2 A4.2 KEYROW . . . 889
39.3 A4.3 MODE . . . 889
39.4 A4.4 The Thor Windowing System . . . 890
39.5 A4.5 BEEP . . . 890

**40 A5 Expansion Boards 891**
40.1 A5.1 GOLD CARD . . . 891
40.2 A5.2 SUPER GOLD CARD . . . 891
40.3 A5.3 AURORA . . . 892
40.4 A5.4 Q40 . . . 892
40.5 A5.5 HERMES / SuperHERMES . . . 893
40.6 A5.6 QuBIDE . . . 893

**41 A6 Compatibility 895**
41.1 A6.1 Addressing . . . 895
41.2 A6.2 Speed . . . 895
41.3 A6.3 The Operating System . . . 896
41.4 A6.4 Memory . . . 896
41.5 A6.5 The Stack Pointer . . . 896
41.6 A6.6 Compilers . . . 896
41.7 A6.7 High Resolution Displays . . . 897
41.8 A6.8 String Lengths . . . 897
41.9 A6.9 Later Processors & Gold Cards . . . 897
41.10 A6.10 Finally . . . 897

**42 A7 Multiple Basics 899**
42.1 A7.1 MINERVA MultiBASICS . . . 899
42.2 A7.2 SMS Multiple SBASICs . . . 901

**43 A8 Error Messages 905**
43.1 A8.1 Standard English Error Messages . . . 905
43.2 A8.2 Foreign Error Messages . . . 908
43.3 A8.3 Dates . . . 910
43.4 A8.4 SMS Messages . . . 911

**44 A9 Character Set, Keyboard 927**

44.1 A9.1 The Character Set . . . 927
44.2 A9.2 Keyboard Layouts . . . 934

**45 A10 Designing New Character Sets (Fonts) 937**
45.1 A10.1 Fonts on the QL . . . 937
45.2 A10.2 Changing Fonts in Programs . . . 937

**46 A11 Mathematics 941**
46.1 A11.1 Degrees and Radians . . . 941
46.2 A11.2 Triangles and Trigonometrics . . . 942
46.3 A11.3 Boolean Logic . . . 943
46.4 A11.4 Operators . . . 944
46.5 A11.5 Hexadecimal and Binary Numbers . . . 946
46.6 A11.6 Integers . . . 946
46.7 A11.7 Faster Mathematics . . . 947
46.8 A11.8 Precision . . . 947

**47 A12 Device Drivers 949**
47.1 A12.1 Devices in General . . . 949
47.2 A12.2 Directory Device Drivers . . . 950
47.3 A12.3 Window Device Drivers . . . 953
47.4 A12.4 Other Device Drivers . . . 955
47.5 A12.5 DIRECT SECTOR ACCESS . . . 973
47.6 A12.6 Level-1 Device Drivers . . . 974
47.7 A12.7 Level-2 Device Drivers . . . 974
47.8 A12.8 Level-3 Device Drivers . . . 975
47.9 A12.9 Using Alien Format Disks . . . 976

**48 A13 Extended Pointer Environment 977**

**49 A14 Coercion 979**

**50 A15 Mouse Drivers 981**
50.1 A15.1 A Mouse for the Standard QL . . . 981
50.2 A15.2 A Mouse for QPC / QXL . . . 983
50.3 A15.3 A Mouse for ATARIs . . . 983
50.4 A15.4 A Mouse for Unix and Macintoshes . . . 983
50.5 A15.5 A Mouse for the Amiga . . . 983

**51 A16 The QL Display 985**
51.1 A16.1 The Screen Address . . . 985
51.2 A16.2 The Screen Size . . . 985
51.3 A16.3 On-Screen Colours . . . 986
51.4 A16.4 USING HIGH RESOLUTION DISPLAYS . . . 1021

**52 A17 Networks 1023**
52.1 A17.1 QNet . . . 1023
52.2 A17.2 Flexynet (DIY Toolkit - VOL X) . . . 1027
52.3 A17.3 Midinet . . . 1028
52.4 A17.4 Sernet . . . 1028
52.5 A17.5 Amadeus Interlink . . . 1029
52.6 A17.6 QL - PC Fileserver . . . 1029

**53 18 Configuring Programs** **1031**
53.1 CONFIG Level 1 & Level 2 . . . . . 1031
53.2 Passing Parameter with EXEC . . . . . 1031
53.3 Making the configuration part of the program. . . . . 1032
53.4 Using a separate configuration file. . . . . 1032
53.5 Using Environment Variables . . . . . 1032
53.6 DATA_USE etc . . . . . 1032

Contents:

CHAPTER

# ONE

## ORIGINAL FOREWORD

We first met Rich Mellor at the QL Club International meeting in the Midlands in 1995. I had read a lot of the articles and reviews that he had published in the, now defunct, QL World and, as is usual when you know someone through the written word alone, I was unprepared for the man that I met.

It was not until I took down his name and address when he bought some software from us that I found out who this young and enthusiastic QLer was. During the course of the day he sat down with and discussed the Flashback database system which Steve Hall had been running under SMSQ/E. We were trying to get this program adapted to run properly under SMSQ/E and Rich offered to help do that. He proved to be a formidable associate. Having once set a target in his sights he nagged away at it until he achieved his purpose and in the course of doing that uncovered a numbers of bugs and inconsistencies in the behaviour of SMSQ/E.

Since we received very little co-operation from the Flashback side of things we have, sadly been unable to bring the product back onto the market place. In the meantime, Rich has managed to vastly improve the program and now sells this new version as an upgrade to the original Flashback SE program.

About twelve months after meeting Rich, we were offered a route finding program written in Atari Basic. We passed the project to Rich who attacked it with his usual vigour and soon was assaulting us with disks of Beta versions. Q-Route was born in 1997 and has proved a very popular program, with the latest version making use of the Extended Colour Drivers available under the latest versions of SMSQ/E.

It was at this point that he first mentioned this book. The bulk of the work here is his, although there have been contributions in the past from Franz Hermann and Peter Jaeger (both of whom have given their kind permission for the work to be published). It became obvious as I was printing this that we could not do it all in one volume and also that we had to provide it in loose leaf form, instead of as a bound book, so that we could easily provide updates as they become necessary.

The book arrived at the QBranch Headquarters on floppy disks and my EPSON printer worked overtime printing the final version. It was produced entirely using QL Software and Hardware. The copy was generated in Text 87 plus4 on a Minerva QL / Super Gold Card running under SMSQ/E. The final copy here was printed on an Aurora / Super Gold Card / SuperHermes / RomDisq machine to either an EPSON Stylus 200 Inkjet or an EPSON Stylus 850 Inkjet printer and the result passed to a local printers to produce the copy that you now hold. I hope that you will find this useful and informative and that it will inspire you to produce some elegant programming for the QDOS / SMSQ community.

Since producing the original version of this Manual, Rich has released Q-Help together with a broad range of other software under the guise of RWAP Services. Q-Help provides the basic detail of all of the keywords covered in this book, together with their syntax in an easy to use program. This can be linked with the Q-Index program which is supplied with this book to form the ultimate cross-reference guide to the QL's BASIC. Q-Help forms a welcome companion to this manual and is especially useful if you are travelling. The Sinclair Spectrum is now enjoying something of a revival, following the re-release of its games for use with the Amstrad E-m@iler phone. The future of the QL is also looking increasingly rosy

(well at least colour-wise), since the release of SMSQ/E v2.98+ which supports up to 16 million colours on the Q40, QXL and QPC2 systems. The day when the QL can access the internet and email is also now at hand. We hope that this will keep people more interested in the QL and its software.

We shall attempt to keep the manual up to date as much as possible with current developments in the QL world, as new emulators and QL compatibles are released. If you have any further information which you feel should appear in this Manual, then please let us have full details in order that it may be of benefit to the whole QL community.

I am sure we will hear more of Rich Mellor in the future.

*Roy Wood, Q Branch HQ, Portslade, Sussex 2002.*

## 1.1 2015 Foreword

Since the above Foreword was written, the SBASIC/SuperBASIC Reference Manual has been enhanced through 4 different update releases, and proved a best seller despite the 2 large A4 volumes with over 1000 pages of information on the QL and its SuperBASIC.

Since then, the Manual was re-released in PDF format and has now, in recognition of the 30th Anniversary of the launch of the Sinclair QL, been converted to HTML and put onto the internet to be updated as a QL Community Project.

I still continue to support the Sinclair QL, its emulators and clones, ensuring a constant supply of parts and second hand items through my own website www.rwapsoftware.co.uk[1] and through my own retro and vintage computer trading website www.sellmyretro.com[2]

The manual appears here in all its glory, with the bulk of the conversion having been automated, but awaiting community input to improve the layout and add new entries to the Manual.

*Rich Mellor, RWAP Software, Stoke-on-Trent, 2015*

## 1.2 Online Edition Foreword

When Rich put the HTML version online and asked for volunteers to tidy it up, maintain it etc, lots of people *didn't* step forward! I was one of the ones who did and spent many a happy lunch hour downloading pages, stripping out the invalid HTML, adding paragraph tags, converting listings to the correct format etc etc. The results were uploaded by Rich and a nicer, or at least, tidier, version began to take place.

This colourful version is the result of some *playing* with a product called **Sphinx-doc** on my Linux box. It takes files in the format of Restructured Text - basically, a plain text file containg your data and markup commands - and from those, builds all sorts of output such as epub, pdf and the HTML you are reading right now.

I hope you like it. It takes a long time to get it just right.

You can see the original source for each page by clicking the link 'View page source' at the top right of each page.

*Norman Dunbar, Dunbar IT Consultants Ltd, Leeds, West Yorkshire, 2015/2016*

[1] http://www.rwapsoftware.co.uk
[2] http://www.sellmyretro.com

CHAPTER

# TWO

# INTRODUCTION

The Sinclair QL was officially released during 1984, and since that date, has gone through several changes to both the hardware and the operating system. Unfortunately, when the rights to the QL were sold to Amstrad it looked as if the end of the QL was near in view of the fact that Alan Sugar, Managing Director of Amstrad decided that the QL should be withdrawn from the market.

Since that fateful day, several types of QL replacement have emerged, including new Hardware platforms such as the THOR range of computers, the AURORA replacement motherboard, the ST/QL hardware emulators (including QVME and the Mode 4 Emulator), the Q40 and the QXL, a hardware emulator that can be plugged into a PC. Several software emulators have also been developed, allowing QL software to run on PCs, Amigas, Apple MacIntosh and Unix based computers.

Some of these emulators are much faster than the original QL computer, which itself can be speeded up by the use of new operating systems (SMSQ/E and Minerva) and new, faster expansion boards (Gold Card and Super Gold Card).

The QL has also been further expanded by the ability of the emulators and the new AURORA and Q40 motherboards to handle much higher resolutions (and more colours in the case of AURORA and Q40).

The QL is different from several other more popular computers in that it has a built-in programming language (SuperBASIC) which has survived (mainly unchanged) since the QL was first released. Various assertions were made concerning the abilities of SuperBASIC when the QL was first launched many of which did not exist at the time. Most of those promises have now been fulfilled by third party toolkits. There are now even two multitasking versions of SuperBASIC, called MultiBASIC which is built into Minerva and SBASIC which forms part of the SMSQ/E operating system.

We, the authors of this book, think that SuperBASIC is a superb programming language for several reasons:

- SuperBASIC is part of the QL, whose multitasking operating system QDOS (and now also SMSQ/E) is extremely powerful even when compared with more popular (and expensive) ones.
- SuperBASIC was originally implemented as an interpreter. This makes program development very fast. Several compilers are available which allow you to produce programs which can run easily and quickly on all versions the QL.
- The version of SuperBASIC provided with the SMSQ/E operating system is extremely quick; in many cases, faster than when compiled with Qliberator.
- SuperBASIC is designed to be extendible from SuperBASIC itself through user-defined PROCedures and FuNctions as well as from machine code via resident toolkits.

The latter point was indeed the motivation to write this book. There are a lot of really useful toolkits available in the public domain which can be used by a programmer in his or her programs and freely distributed as part of it.

Not only is there a vast range of toolkits available to the SuperBASIC programmer, but the QL's Operating System has also undergone various changes, and with new QL compatible computers and emulators being released, as well as the Minerva replacement ROM and SMSQ/E replacement operating system, it is important that any programmer should know where problems may occur in the use of each SuperBASIC command.

We have therefore attempted to cover each command in sufficient detail, with useful examples, and a commentary on bugs and incompatibilities. No doubt as each new implementation of SuperBASIC comes to light, so will further problems and we will try to keep abreast of these. We would refer you in general to the Appendix on compatibility, which contains various guidelines for ensuring that programs should remain compatible with future operating systems. More specific detail is contained in the description for each command where problems are known to exist.

We have covered the commands contained in the following sources:

- The standard QL ROM © Sinclair Research Limited / Amstrad plc.
- The THOR ARGOS Operating System © CST and DANSOFT
- The Minerva ROM © Minerva and TF Services
- Super Gold Card, Gold Card, QXL and Trump Card © Miracle Systems Ltd.
- SMSQ/E, SMSQ Operating Systems © Tony Tebby
- Toolkit II and Hotkey System II © Tony Tebby
- SERMouse © Albin Hessler Software
- DIY Toolkit (sold as Cardware) © Simon Goodwin
- AtariDOS and ATARI_REXT © Jochen Merz Software
- Turbo Toolkit (freeware) © The Turbo Team, David Gilham & Mark Knight
- DJToolkit 1.16 (freeware) © Norman Dunbar & Dilwyn Jones
- QPC version 4.04 specific commands © Marcel Kilgus.
- As many Public Domain Toolkits as we can find and understand.

We have covered the Toolkit II and Hotkey System II because it is a standard addition to a QL operating system and included on several add-on expansion boards.

As you look at the range of toolkits already available, you will notice that several commands appear in more than one toolkit. Unfortunately, a command with the same name in two toolkits, may in fact have a different syntax or even a different function! It is therefore our hope that with the aid of this book, any future toolkits will remain compatible with earlier ones, and existing toolkits will be amended to resolve these incompatibilities.

If you do come across further incompatibilities, operating system commands or public domain toolkits which are not noted in the manual, then please do contact us as soon as possible so that we may investigate the situation and incorporate them in the book. We may even find ways of correcting the errors!

In the meantime enjoy SuperBASIC !

## 2.1 Contributing Authors

- Franz Herrmann
- Peter Jager
- Rich Mellor
- Norman Dunbar - DJToolkit 1.16 additions.
- Norman Dunbar - QPC 4.04 additions.
- (Norman Dunbar - HTML conversion, tidyup etc.)

## 2.2 Installing Toolkits

Most toolkits can be loaded and linked into the QL's operating system as an addition to the existing SuperBASIC (or SBASIC) keywords simply by using a command similar to one of the following three examples:

```
LRESPR flp1_Toolkit_bin
```

```
A=RESPR(x): LBYTES flp1_Toolkit_bin, A: CALL A
```

```
B=ALCHP(x): LBYTES flp1_Toolkit_bin, B: CALL B
```

(where x is the length of the toolkit code).

Some toolkits include additional device drivers and must be loaded into the resident procedure area of memory and therefore the third example above must not be used, and the toolkit must be linked in before any jobs are EXECuted (or else the toolkit can crash your system). We have tried to include a reference in this book where this is the case.

The normal sequence of events for loading toolkits and other extensions to the operating system is set out below:

1. Link in any speed enhancements (such as Lightning or Speedscreen) <— not needed on SMSQ/E
2. Load any additional device drivers (such as Mem or History) <— check which ones are already included in SMSQ/E
3. Link in all required toolkits (those which contain device drivers should be linked in first).
4. Load the Pointer Environment (if required) <— not needed on SMSQ/E
5. Load a secondary program to carry on setting up the system - this is because on pre-JS ROMs, any keywords added by toolkits are not available for use in the same program which linked them in.
6. Start up any required Jobs (such as ALTKEY, FSERVE or the Buttons provided by the Pointer Environment).
7. Use HOT_GO if you use the Hotkey System II.

However, some toolkits insist that you enter a command before you can actually use any of the other keywords provided by that toolkit. The following toolkits need this:

- Toolkit II- You will need to enter the command TK2_EXT, unless Toolkit II is built into your operating system (such as SMSQ/E) or you have used the commands AUTO_TK2F1 or AUTO_TK2F2

- BeuleTools- You need to enter the command Beule_EXT
- BTools- You need to enter the command BTOOL_EXT
- Tiny Toolkit- You need to enter the command TINY_EXT
- ATARI_REXT- You need to enter the command ATARI_EXT
- Hotkey System- You need to enter the command HOT_GO for any of the ALTKEY (or other HOT_xxx) keywords to work.

(See the individual commands listed above for further details).

CHAPTER

THREE

# CREDITS

A lot of the information contained in this manual has been based on the original documentation provided with the toolkits, some authors having written extensive manuals. The QL User Guide (an excellent introduction to programming in general and SuperBASIC), the Toolkit II, SMSQ/E and Minerva manuals and the documentation of the Public Domain Math Package must also be explicitly mentioned for their quality. You will seldomly - we dare say almost never - find a good program with poor documentation.

Great respect must be paid to Roy Atherton, Stephen Berry, Tony Tebby, Laurence Reeves and Helmut Aigner.

Simon Goodwin has developed, with the help of readers, many extensions to SuperBASIC in his popular DIY Toolkit column in the Sinclair QL World magazine. The bundled extensions are now available from Public Domain libraries as cardware (please send Simon a postcard if you find the routines useful). We would like to thank Richard Alexander (formerly of C.G.H. Services) for providing us with a copy of DIY Toolkit and his encouraging support. Over the years, the DIY Toolkit series has inspired many people to write their own public domain toolkits. Simon Goodwin has also revealed many bugs in the original QL keywords in QL World articles.

Details on selected items have been received from (in alphabetical order)

- Tiago Leal,
- Thomas Menschel,
- Mike Panagiotopoulas,
- Peter Recktenwald,
- Laurence Reeves,
- Andreas Rudolf,
- Bernhard Scheffold,
- Peter Sulzer,
- Kees van der Wal,
- Dave Walker
- and more.

Thanks to Boris Jakubith who allowed the inclusion of his ingenious History Device.

Special mention must also be made of Q Branch and Jochen Merz for their support in resurrecting this project.

This manual deals with the current versions of SuperBASIC, if you want to use machine code to extend SuperBASIC further, we highly recommend the 'QDOS / SMS Reference Manual' available from Jochen

Merz software and Q Branch. The technical details of SuperBASIC are contained in 'QL SuperBASIC The Definitive Handbook' by Jan Jones, available from Quanta.

## 3.1 Other Notices

- Minerva and Hermes are products from QView and T.F. Services. Minerva MKII and related interfaces are also products from this source.
- Sinclair, QL, QDOS, QNet, ZX81 and ZX Spectrum are trademarks of Sinclair Research Ltd.
- Supercharge, Turbo, Toolkit 3, Conqueror and Solution are products from Digital Precision Ltd.
- QLiberator is a product of Liberation Software.
- QL World magazine was published by Arcwind Ltd.
- QXL, Super Gold Card, Gold Card and Trump Card are products from Miracle Systems Ltd (now available from Q Branch).
- Extended Pointer Environment, Toolkit II, QPAC2 and Config are products from Qjump (Tony Tebby) and Jochen Merz Software.
- Atari QL-Emulators (ST/QL) and the SMSQ/E operating system are products from Jochen Merz Software.
- THOR computers were a product of Dansoft.
- The word processors Perfection and Text87Plus4 have been used to write this book.
- The QED public domain editor from Jan Bredenbeek and the named pipes driver from Hans Lub (and built into SMSQ/E) have also proved to be very helpful.
- DEAssembler and MasterBasic from Ergon Software were also used in the production of this book.
- QLs (and compatibles) were of course entirely used.

For the online version, Linux played a big part. It was responsible for:

- Downloading the original HTML files;
- Cleaning them up, as much as possible, using HTMLTidy;
- More automated cleaning using the wonderful sed utility;
- Conversion from (clean) HTML to Restructured Text using pandoc;
- Conversion of the Restructured Text files into any output format you could possibly desire, using Sphinx-doc.

CHAPTER

FOUR

# STRUCTURE OF THIS BOOK

This book consists basically of three parts.

The first part forms those obligatory sections, from *Introduction* to *Writing Programs*.

The second part is the main part which is explained below.

Finally the *Appendices* make up the third part. These appendices have been added to prevent repeated explanations in the main section, they are not a full-blown concepts section.

The main part of this book is the *Keywords* section. This section is sorted alphabetically and for each keyword there will appear (at least) a description of the keyword's syntax, where it can be found and a short description. In most cases, you will also find examples and cross-references to other keywords, and from time to time some notes on using the commands. You may even come across warnings.

The alphabetical list is arranged in the following order (it is not case sensitive):

ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 % $ _

This means for example that the keywords such as S_LOAD appear at the end of all other keywords beginning with S.

## 4.1 Syntax

Compressing all possible variations of a keyword's syntax in usually one abstract line can be difficult for those readers who are not familiar with syntax schemes. That's why we want to explain our notation in detail.

Throughout the book, almost everything that can be typed into the computer or returned by it, is written in a different typeface (italics) so that you can always easily distinguish those parts of the text which can be entered into the computer. We have tried to be as consistent as possible in this respect.

The syntax scheme itself contains symbols which are not to be typed in and thus appear in the normal typeface:-

### 4.1.1 Square Brackets ([ ])

These indicate that the enclosed parts are optional. Optional parameters (ie. parameters which can be omitted without producing an error) are always in square brackets.

Example:

DLIST [#ch]

Both DLIST and DLIST #2 are valid.

### 4.1.2 Square Brackets With Superscript Asterisks ($^*$[ ]$^*$)

These suggest that the number of optional parameters is not limited, ie. there can be any number of such parameters. Another symbol for the same meaning are nested square brackets with three dots inside.

Example:

POINT x,y [,$x^2$,$y^2$ [,$x^3$,$y^3$, . . . ]]

or

POINT x,y $^*$[,$x^i$,$y^i$]$^*$

Any non-zero number of co-ordinate pairs are allowed. Note that the indices are also symbols, used to make reading easier. Of course you cannot type POINT $x^1$,$y^1$ but just POINT x1,y1 without any subscript characters will work.

### 4.1.3 Curly Brackets ({ })

These mean that the parameter can be chosen from a limited variety of types which are given in the brackets. The options are mutually exclusive and separated by a vertical line (|).

Example:

KEYWORD {test$ | test%}

Either KEYWORD test$ or KEYWORD test% is valid.

The vertical line (|) can also appear in square brackets. In this case, the parameter is optional and has to be selected from one of the types listed in the brackets.

Example:

SIZE test[%|$]

SIZE test%

SIZE test$

SIZE test

are all valid.

We generally assume that you have some basic idea of SuperBASIC syntax because this book is not a SuperBASIC tutorial but a reference book for toolkit keywords.

### 4.1.4 Channels (#ch)

Many Syntax definitions refer to a channel parameter, which is normally shown as #ch or #channel.

These channels can have two main types, a channel connected to a Device (or File) and a channel connected to a Window (a scr_ or con_ device). The type of a channel is specified when that channel is OPENed - see the description of OPEN for further details.

Normally the description for each keyword will specify if the channel used by that keyword has to be of a specific type. If no mention is made, then presume that the keyword can be used on any type of channel.

### 4.1.5 Location

This is just the name of the toolkit(s) where you will find the keyword.

Some locations are not separately available toolkits, eg. QL ROM, Super Gold Card, Gold Card, Trump Card, ST/QL and more.

Where the Location is given as QL ROM, this means that the keyword is available on all versions of the QL, QL compatible computers and Emulators (unless specified).

Where the Location is given as Gold Cards, this covers the whole of the Gold Card range of expansion boards, namely Super Gold Card and Gold Card. However, note that commands given by these boards will not be available under SMSQ/E unless specifically stated.

Some keywords are available as part of the Level-2 or Level-3 device drivers.

Level-2 device drivers are built into Gold Card, Super Gold Card and the QUBIDE disk interface, as well as forming part of SMSQ on the QXL and the ST emulators. Level-2 device drivers are also available separately for the Trump Card.

Level-3 device drivers are provided with SMSQ/E and incorporate all of the features of Level-2 device drivers and more. Therefore if the location is said to be Level-2 Device Drivers, these commands will also work on Level-3 Drivers.

SMSQ/E is a new operating system which is compatible with QDOS and incorporates all of the original QL ROM keywords, Toolkit II, the Pointer Interface, Window Manager, Hotkey System II and Level-3 device drivers. Therefore if a keyword is listed as appearing in any of these, then it will be available to the SuperBASIC programmer under SMSQ/E.

SMSQ is the operating system built into QXL which can be replaced by SMSQ/E. Both operating systems are very similar in how they treat SuperBASIC keywords and we have therefore used to SMS to indicate that a comment may apply to both SMSQ/E and SMSQ. Their individual names have been used if there is a difference.

ST/QL refers to the full range of QL Hardware Emulators for the Atari ST (Extended Mode-4 Emulator, QVME and the original ST-QL Emulator). Any comments which refer specifically to one of the boards are covered separately.

Refer to the Emulators Appendix for more details on the various emulators available.

## 4.2 Description

The description of the function of a keyword is usually abstract and relatively short. You may have to read it carefully to understand it fully. Technical details are limited to the needs of a SuperBASIC programmer, but we document the current standards in QL programming and environments.

## 4.3 Examples

The examples demonstrate the different syntax variations of a keyword and explain concrete usages. We have tried to write some short example programs which make sense outside pure computer applications, meaning that a brief explanation is seldom necessary.

All listings have been directly imported from the SuperBASIC interpreter into the word-processor via an intermediate file or pipe. The exceptional multitasking capabilities of the QL with Tony Tebby's Pointer Environment allowed us to write text, try out toolkits and develop examples, all at the same time. Due to the direct import of the latter, mistakes in the examples have been minimised. However, we are all only human.

It is not our intention to praise a particular style of programming. Book space, layout, typefaces and didactic considerations posed various limits. For this reason, examples or parts of examples which are designed as modules (procedures or functions), will usually not check the supplied arguments for wrong parameters.

All example listings are freely distributable subject to restrictions. You are allowed to develop applications from examples or make use of examples in other programs under the condition that this book and its authors are given credit accordingly.

## 4.4 Notes

These (sometimes extensive) comments vary from strange side-effects of keywords to off-topic remarks. They have been added for completeness. Very often the original documentation did not recognise all possible implementations for practical reasons: a certain configuration did not exist at the time of writing, the author did not expect users to exploit parameter ranges to the full, etc.

It is not necessary to know the notes but when struggling against odd phenomena, reading the notes could clarify seeming mysteries.

## 4.5 [Implementation] Notes

When bringing out new implementations of the QL ROM, the authors are limited by the amount of memory into which they have to squeeze all additions, modifications and corrections. They therefore tend to extend the syntax instead of adding new keywords. That is why the Implementation Notes are usually a further description of syntax and usages, possibly including examples. POKE is a good example.

The more common Implementation Notes are for Minerva, THOR and SMS. Please note that throughout references to SMS refer to both the SMSQ and SMSQ/E operating systems (see above).

Implementation Notes may also appear for each of the different Emulators and Expansion Boards which can be used with the QL.

## 4.6 Warning

An absolutely obligatory section! Some commands and functions crash the machine under certain circumstances: the warnings are intended to help you avoid frustration and disappointment. Please do not blame the authors of the toolkits for the bugs, writing a fool-proof program is very time consuming and nobody is perfect (neither the toolkits' authors nor the writers of this book). If we forget to mention a dangerous situation, this is because we were not aware of it.

## 4.7 Cross-Reference

Keywords can be connected by a couple of links. They can do almost the same or perform similar functions, in these cases we did not make use of the word-processor's block copying facilities to artificially enlarge the book but simply referred to another text passage. If the relationship between keywords is emphasised by their name, cross-references may be extremely short or missing; due to the alphabetical order of the keywords, the reference is not too far away in most cases anyway.

Cross-references may also give notice of other keywords where the relation is rather indirect, this has been done to encourage liberally skipping through the pages.

CHAPTER

FIVE

# WRITING PROGRAMS

There have been many books and magazine articles written about how to write SuperBASIC programs, so we do not intend to cover the basic principles here. The main section detailing the various keywords available to the SuperBASIC programmer contains many useful examples and we suggest that you work through those examples, making sure that you understand how they work and trying to improve them if possible.

In this section, we look at some of the major problems which can face the SuperBASIC programmer and hope to provide some guidance as to how you can overcome those problems and ensure that the programs which you write can be used successfully on all implementations of the QL's operating system.

## 5.1 Compiling SuperBASIC Programs

Digital Precision's Supercharge and Turbo compilers are not able to compile some keywords` (eg. those which allow arrays as parameters) while Liberation Software's QLiberator can compile every additional keyword which makes sense in a compiled program. Although programs compiled with Qliberator will be slower than those compiled with Turbo, the fact that Turbo has not been updated for a number of years (and still contains certain bugs) means that Qliberator may be a better option. This is a matter of fact which we consider worth mentioning for the benefit of Supercharge and Turbo users, it is not intended as a hidden advertisement for QLiberator.

**Note**

While the above was certainly true when the first paper version of this manual was printed, progress has been made, as the following correction from George Gwilt explains:

For versions of TURBO earlier than 4.21 machine procedures or functions that modify their parameter values, process arrays (other than single strings), manipulate the stored program text, or rely on other interpreter data structures (such as the name table and name list) will not work when compiled. The majority of add-on commands do not do this, and consequently work perfectly.

For TURBO v4.21 and later none of these restrictions apply except for the reliance on the name list.

On the other hand, if you are a Minerva freak, use the Pointer Environment in your programs or want to ensure that your programs will run on SMSQ/E, then you might just find that QLiberator is the better compiler.

As you look through the book, you will find that many keywords act differently on the various implementations of SuperBASIC. To overcome this problem, we suggest that programs which are designed to run on various implementations of the QL, should be compiled, as subject to the comment below,

Compilers ensure that programs will run on any system (provided that all keywords used by the program are available on that implementation).

If you want to write programs which use built-in functions only if they are available, you will need to use Qliberator which does not report an error until you try to use an undefined keyword (Turbo and SuperCharge both refuse to load the program in this instance).

Together with the use of the VER$ function, you can easily write programs which work on all implementations of the QL making use of extra facilities which may be available on some versions of the operating system.

One thing of which you must be careful when compiling programs to run on other implementations is the various notes and warnings given for some keywords. Some keywords cannot be compiled and others may have bugs on various implementations which are not fixed by the compiler (refer to the compiler's manual to see if they rectify the bugs). One of the main culprits of this is the CURSOR command which will reject the use of five parameters on pre-MG ROMs.

Turbo does tend to include its own code to overcome any incompatibility problems with standard QL ROM keywords, whereas Qliberator tends to use the native routines on the operating system on which it is running.

Another thing to bear in mind when writing programs for use with a compiler is that you should really ensure that the program opens all of its own windows and does not assume that any channels are already open. You will also need to remember to DIMension any strings used by the program to ensure that the program works the same way as it does under the Interpreter when compiled (see DIM for an explanation of the way in which strings are treated in various circumstances).

It is also useful to include your own error trapping routines in a compiled program (such as WHEN ERRor) - most compiler error messages are very unhelpful when seen by a user of a program and in particular, there is a problem with programs compiled with Turbo and Supercharge in that they do not wait for the user to press a key after reporting an error before stopping the compiled program. This is fine on a standard QL, as the final display of the program is left on screen - however, under the Pointer Environment, unless the program is started with the command EXEP flp1_test_obj,u (or similar), then the Pointer Environment removes every last trace of the program from the screen when it stops. Also, if any machine code Procedures or Functions report an error, then the error may not be reported and the program may just 'hang' if #0 is not open.

One of the main problems with compiled programs is the Cache provided with 68020 processors (and faster). Caches cause problems with machine code which modifies itself (normally to enhance speed). Whereas programs compiled with Qliberator should be okay (depending on the toolkits used within them), TURBO compiled programs are normally self- modifying. However, provided that a program is compiled under TURBO with the BRIEF directive then it will run provided that the cache is switched off for the first 0.3 seconds after EXECing the program - for example, if a program causes problems, use:

```
10 CACHE_OFF
20 EX flp1_PROGRAM_exe
30 PAUSE 15
40 CACHE_ON
```

Refer to CACHE_ON and CACHE_OFF for further details.

One of the other differences between TURBO compiled programs and Qliberator compiled programs is that the former all make assumptions about the start of the screen and system variables. However, there is a public domain program by Davide Santachiara called Turbostart which resolves these problems by altering compiled programs.

## 5.2 Writing Programs to Run Under the Pointer Environment

It is not as difficult as it first may seem to write programs to run under the Pointer Environment, unless you intend your program to use the Pointer and Menu facilities provided by the Pointer Environment.

Basically, any program which has been written with the intention of multitasking will work under the Pointer Environment. However, the program should not attempt to tie up the QL's resources unless it is using them (for example, do not open a printer channel until you need to send output to it) and then close the channel once all output has been sent. It is also useful to allow the user to add their own facilities such as a mouse through amending the boot program.

A SuperBASIC program will of course only multitask if it is compiled (see Section 4.1) or if the user has Minerva or SMSQ/E which provide multitasking SuperBASIC interpreters.

### 5.2.1 Using the Pointer and Menu Facilities

If you intend to use the Pointer and Menu facilities provided by the Pointer Environment, then you will need to use either QPTR (from Jochen Merz Software) or EasyPTR (also from Jochen Merz Software). The latter is easier to get to grips with and use from SuperBASIC, but is less flexible than QPTR. You may also want to use QMenu (also from Jochen Merz Software) which provides various ready made menus which can be easily added to BASIC programs in order to provide standard facilities such as getting the user to select an existing filename.

Your program will need to set itself an Outline (see OUTLN) and also check on the screen display size (see SCR_XLIM and SCR_YLIM). You may also want to check that the Pointer Environment is available (see P_ENV).

If a program does not define an OUTLN properly, then you may notice that some parts of the program's display disappears - the reason for this is that when the program is first loaded, the Pointer Environment uses the OUTLN of the calling Job to define the maximum size of the windows which the program may use - this may be too small and your own program should therefore define its own OUTLN.

If the OUTLN setting is too small, you may notice that some EasyPTR menus will not appear on screen - this is because if you try to OPEN a window which appears partly outside the OUTLN setting, then that window will be OPENed to be the same size and position as the OUTLN setting. If you try to use WINDOW to position an existing window so that any part of it would fall outside of the OUTLN setting, then an error will be reported.

Other problems will occur if you CLOSE the window which has the OUTLN defined - the OUTLN will become the smallest area possible which encompasses all currently OPEN windows - and will become attached to the smallest existing channel number - this unfortunately means that the contents of any windows which have been CLOSEd where those windows (or part of them) fall outside the new OUTLN, will disappear!! This can result in some programs losing parts of their display.

An example of this can be seen with the program:

```
5 OPEN #0,con
10 OUTLN #0,448,200,32,16:PAPER #0,0:CLS#0:INK #0,4:PRINT #0,'This is #0'
20 OPEN #1,con_400x160a40x40:PAPER 2:CLS:INK 7:PRINT 'This is #1'
30 OPEN #2,con_300x100a80x70:PAPER#2,7:CLS#2:INK#2,2:PRINT #2,'This is #2'
40 PAUSE #0
50 CLOSE #0
55 PAUSE #1
60 OPEN #0,con_448x200a32x16:PRINT #0,'This is #0'
```

Try compiling this program as flp1_test_obj and then enter the command EX flp1_test_obj - see what happens when #0 is CLOSEd?

Compare the result if you changed line 50 to CLOSE #2.

One of the solutions to this problem is to use the G option on the EXEP command to define a Guardian Window - try the command: EXEP flp1_test_obj,g,512,256,0,0. The other answer is to OPEN another channel (for example #3) to be the OUTLN channel before any windows are OPENed - change line 5 and 10 thus:

```
5 OPEN #3,con_:OUTLN #3,448,200,32,16
10 OPEN #0,con_448x200a32x16:PAPER #0,0:CLS#0:INK #0,4:PRINT #0,'This is #0'
```

These problems with defining OUTLN's will become apparent if you test programs under an SBASIC interpreter and then after the program has been compiled - if you use the SBASIC command to start up a Multiple BASIC (or similar on Minerva) and then LRUN your BASIC program, the OUTLN is always set to OUTLN 512,256,0,0 whereas if you EXECute a compiled program (or even if you use a command such as EX flp1_program_bas to start up a BASIC program under SMSQ/E), the OUTLN will be that set in the calling program (unless defined in the program itself).

## 5.3 Multitasking Programs

If you write a program which is to run under the Pointer Environment, it is useful to remember some rules:

- There is no need to activate the cursor on the program - when the program is PICKed by the user, then any open con_ channel is automatically activated. You may however, still wish to do this if the program is to be able to run without the Pointer Environment.
- If any part of the job's OUTLN is overlapped by other programs, then the job will not be able to access and scr_ or con_ channels (it will wait until the program is activated). This can be overcome with PIE_ON / PEON or by starting the program with EXEP (using the U parameter). You can check if a program can write to a screen channel with PEND.
- As soon as the program ends (with STOP or RJOB) then all of its windows will be removed from the screen, again unless you have used EXEP with the U parameter.

CHAPTER

SIX

# KEYWORDS INTRODUCTION

The following is a general description of the various commands available to the SuperBASIC programmer, as provided by Public Domain toolkits, QL Emulators and SuperBASIC itself.

We have indicated which of the commands are functions (and therefore must appear in a program in the form):

```
x = FUNCTION ( parameter )
```

or:

```
IF FUNCTION ( parameter ) = value
```

and those which are procedures (also known as commands) which must therefore appear in a program in the form:

```
PROCEDURE parameter
```

**NOTE**

Many toolkits insist that you initialise the toolkit before you can use the various keywords contained in those toolkits. Refer to Section 1.1 for details.

CHAPTER

SEVEN

# TOOLKITS

This manual covers a lot of toolkits, and there are probably many more to be added as time passes. This page of the manual lists the known toolkits in the manual at present (so will obviously need to be updated as times goes on) and shows simple links to the pages holding details of the commands provided by those toolkits.

If more than one toolkit provides a command with the same name, the link will be to the *first* known entry in the manual. The page linked to should contain all versions of the particular command.

## 7.1 Ähnlichkeiten

The commands in this toolkit are:

- *PHONEM*
- *SOUNDEX*
- *WLD*

## 7.2 ARRAY

The commands in this toolkit are:

- *LAR*
- *SAR*
- *SARO*
- *SEARCH*
- *SORT*

## 7.3 ATARI Emulators

The commands in this toolkit are:

- *MIDINET*
- *MNET*
- *MNET_OFF*
- *MNET_ON*
- *MNET%*

- *MNET_S%*
- *MNET_USE*
- *SERNET*
- *SNET*
- *SNET%*
- *SNET_ROPEN*
- *SNET_S%*
- *SNET_USE*

## 7.4 ATARIDOS

The commands in this toolkit are:

- *ACOPY*
- *ADELETE*
- *ADIR*
- *AFORMAT*
- *AQCONVERT*
- *ASTAT*
- *IQCONVERT*
- *QACONVERT*
- *QCOPY*
- *QICONVERT*

## 7.5 ATARI_REXT

The commands in this toolkit are:

- *A_SDATE*
- *A_SPEED*
- *EXCHG*
- *EXTRAS_W*
- *KBD_RESET*
- *ROM_EXT*
- *WSET_DEF*
- *WSET*
- *PEEK$*
- *POKE$*

### 7.5.1 ATARI_REXT - Pre v1.21

This version of the toolkit provides the following, additional, commands:

- *ROM_LOAD*

### 7.5.2 ATARI_REXT - v1.21

This version of the toolkit provides the following, additional, commands:

- *EPROM_LOAD*

### 7.5.3 ATARI_REXT - v1.24 to v2.15

This version of the toolkit provides the following, additional, commands:

- *SND_EXT*

### 7.5.4 ATARI_REXT - v1.29

This version of the toolkit provides the following, additional, commands:

- *XLIM*
- *YLIM*

### 7.5.5 ATARI_REXT - v2.10

This version of the toolkit provides the following, additional, commands:

- *A_RDATE*

### 7.5.6 ATARI_REXT - v2.12

This version of the toolkit provides the following, additional, commands:

- *OUTLN*

### 7.5.7 ATARI_REXT - v2.15

This version of the toolkit provides the following, additional, commands:

- *ATARI_EXT*

### 7.5.8 ATARI_REXT - v2.17

This version of the toolkit provides the following, additional, commands:

- *PEEKS_L*

### 7.5.9 ATARI_REXT - v2.22

This version of the toolkit provides the following, additional, commands:

- *A_EMULATOR*
- *A_MACHINE*
- *A_PROCESSOR*

### 7.5.10 ATARI_REXT - v2.25

This version of the toolkit provides the following, additional, commands:

- *SCR_BASE*
- *SCR_LLEN*

### 7.5.11 ATARI_REXT - v2.27

This version of the toolkit provides the following, additional, commands:

- *A_OLDSCR*

### 7.5.12 ATARI_REXT for QVME - v2.31

The commands in this toolkit are:

- *FREE_FAST*
- *LRESFAST*
- *RESFAST*

## 7.6 Amiga QDOS - v3.20

The commands in this toolkit are:

- *BUTTON%*
- *PTR_LIMITS*
- *PTR_MAX*
- *PTR_OFF*
- *PTR_ON*
- *PTR_POS*
- *X_PTR%*
- *Y_PTR%*

## 7.7 BGI

The commands in this toolkit are:

- *VG_HOCH*
- *VG_LOAD*
- *VG_PARA*
- *VG_PRINT*
- *VG_RESO*
- *VG_WIND*

## 7.8 BIT

The command in this toolkit is:

- *BIT%*

## 7.9 BTool

The commands in this toolkist are:

- *ALCHP*
- *ASK*
- *BASIC*
- *BASIC_F*
- *BASIC_L*
- *BCLEAR*
- *BREAK*
- *BREAK%*
- *BTool_EXT*
- *BTool_RMV*
- *CBASE*
- *CCHR$*
- *CHANID*
- *CHANNELS*
- *CLCHP*
- *CLOSE*
- *CLOSE%*
- *CLRMDV*
- *ConvCASE$*
- *COPY_B*
- *COPY_L*
- *COPY_W*
- *CTAB$*
- *CURSOR*
- *CURSOR%*
- *CVF*
- *CVI%*
- *CVL*

- *CVS$*
- *DEFAULT*
- *DEFAULT$*
- *DEFAULT%*
- *DEFINED*
- *EQ$*
- *ETAB$*
- *EXTRAS*
- *FDAT*
- *FGETB*
- *FGET$*
- *FGETF*
- *FGETH$*
- *FGETL*
- *FGET%*
- *FILE_OPEN*
- *FIND*
- *FLEN*
- *FNAME$*
- *FPOS*
- *FPOS_A*
- *FPOS_R*
- *FPUTB*
- *FPUT$*
- *FPUTF*
- *FPUTL*
- *FPUT%*
- *FREAD$*
- *FREE*
- *FREEZE*
- *FREEZE%*
- *FSETH$*
- *FTYP*
- *FUPDT*

- *FWRITE$*
- *FXTRA*
- *GT$*
- *INPUT$*
- *IO_PEND%*
- *JobCBS*
- *KJOB*
- *KJOBS*
- *MKF$*
- *MKI$*
- *MKL$*
- *MKS$*
- *ODD*
- *OFF*
- *ON*
- *PEEK$*
- *PEEK_F*
- *POKE$*
- *POKE_F*
- *QDOS$*
- *QRAM$*
- *RECHP*
- *RELJOB*
- *REPLY*
- *REPORT*
- *RESET*
- *RJOB*
- *SEARCH*
- *SIGN*
- *SINT*
- *SPJOB*
- *SUSJOB*
- *TPFree*
- *TYPE*

- *TYPE_IN*
- *UINT*
- *WMAN$*
- *XCHANGE*

## 7.10 BeuleTools

The commands in this toolkit are:

- *ALT*
- *ATARI*
- *BAT*
- *BAT$*
- *BAT_USE*
- *BCLEAR*
- *Beule_EXT*
- *BLD*
- *BVER$*
- *CAPS*
- *CLS_A*
- *DBL*
- *EL*
- *ENL*
- *ESC*
- *FF*
- *KEY_ADD*
- *KEY_RMV*
- *KILL*
- *KILL_A*
- *KILLN*
- *LINT2*
- *LMAR*
- *LPOLL*
- *LPR_USE*
- *LSCHD*
- *MD*

- *NIX*
- *NOCAPS*
- *NORM*
- *NRM*
- *PAGDIS*
- *PAGLEN*
- *PAGLIN*
- *PRO*
- *QuATARI*
- *RAMTOP*
- *RESET*
- *RMAR*
- *ROMs*
- *SCREEN*
- *SI*
- *UNL*
- *WIPE*

## 7.11 COMPICT

The commands in this toolkit are:

- *COMPRESS*
- *EXPAND*
- *FASTEXPAND*

## 7.12 CONCAT

The command in this toolkit is:

- *CONCAT*

## 7.13 CONVERT

The command in this toolkit is:

- *CONVERT*

## 7.14 CRYPTAGE

The commands in this toolkit are:

- *LOCK*
- *UNLOCK*

## 7.15 DESPR

The command in this toolkit is:

- *DESPR*

## 7.16 DEV device

The commands in this toolkit are:

- *DEV_LIST*
- *DEV_NEXT*
- *DEV_USE*
- *DEV_USE$*

## 7.17 DIY Toolkit

DIY Toolkit is supplied in a number of volumes, each dealing with a different area of the QL and QDOS. The volumes known to this manual are as follows:

### 7.17.1 Volume A - Alias

The commands in this volume are:

- *_DEF$*
- *_DEF%*
- *_NAME$*
- *ALIAS*
- *CODEVEC*
- *INVERSE*

### 7.17.2 Volume B - Basic Tools

The commands in this volume are:

- *BPEEK_L*
- *BPOKE_L*

### 7.17.3 Volume C - Channels

The commands in this volume are:

- *CHAN_L%*
- *USE*

### 7.17.4 Volume E - Error Control

The commands in this volume are:

- *CHECKF*
- *CHECK%*
- *EDLINE$*
- *PICK$*
- *PURGE*

### 7.17.5 Volume F - File Tools

The commands in this volume are:

- *GetHEAD*
- *SetHEAD*

### 7.17.6 Volume G - Graphics

The commands in this volume are:

- *DRAW*
- *PIXEL%*
- *PLOT*

### 7.17.7 Volume H - Heap and Horology

The commands in this volume are:

- *DISCARD*
- *LINKUP*
- *RESERVE*
- *T_COUNT*
- *T_OFF*
- *T_ON*
- *T_RESTART*
- *T_START*
- *T_STOP*

### 7.17.8 Volume I - Serial Mouse

The commands in this volume are:

- *SYNCH%*
- *X_PTR%*
- *Y_PTR%*
- *BUTTON%*
- *PTR_FN%*
- *PTR_INC*
- *PTR_KEY*
- *PTR_LIMITS*
- *PTR_MAX*
- *PTR_OFF*
- *PTR_ON*
- *PTR_POS*

### 7.17.9 Volume J - Jobs

The commands in this volume are:

- *LIST_TASKS*
- *PRIORITISE*
- *RELEASE_TASK*
- *REMOVE_TASK*

### 7.17.10 Volume M - MultiBASIC

The commands in this volume are:

- *RELOAD*
- *REMOVE*
- *RESAVE*
- *SCR_SAVE*
- *UNLOAD*

### 7.17.11 Volume P - Pipes and Parameters

The commands in this volume are:

- *PARHASH*
- *PARNAME$*
- *PARSEPA*
- *PARTYPE*

- *QCOUNT%*
- *QLINK*
- *QSIZE%*
- *QSPACE%*
- *UNSET*

### 7.17.12 Volume Q - Queues and QDOS

The commands in this volume are:

- *CHBASE*
- *QUEUE%*
- *SYSBASE*

### 7.17.13 Volume R - Replace

The commands in this volume are:

- *LOOKUP%*
- *LOWER$*
- *NEWCHAN%*
- *REPLACE*
- *UPPER$*

### 7.17.14 Volume S - Qlipboard

The commands in this volume are:

- *CLIP$*
- *CLIP%*

### 7.17.15 Volume T - Traps

The commands in this volume are:

- *ADDREG*
- *BTRAP*
- *DATAREG*
- *MTRAP*
- *QTRAP*

### 7.17.16 Volume U - Environment Variables

The commands in this volume are:

- *ALTER*
- *SET*

### 7.17.17 Volume V - More

The command in this volume is:

- *MORE*

### 7.17.18 Volume W - Windows

The commands in this volume are:

- *SET_GREEN*
- *SET_RED*
- *W_CRUNCH*
- *W_SHOW*
- *W_STORE*
- *W_SWAP*
- *W_SWOP*

### 7.17.19 Volume X - MSearch and Vocab

The commands in this volume are:

- *SEARCH_MEM*
- *MSEARCH*
- *VOCAB*

### 7.17.20 Volume Y - FlexyNet

The commands in this volume are:

- *NETBEEP*
- *NETPOLL*
- *NETRATE*
- *NETREAD*
- *NETSEND*
- *NETVAR%*

### 7.17.21 Volume Z - Array Search

The commands in this volume are:

- *INARRAY%*
- *MAXIMUM*
- *MAXIMUM%*
- *MINIMUM*
- *MINIMUM%*

## 7.18 Djtoolkit v1.16

The commands in this toolkit are:

- *ABS_POSITION*
- *BYTES_FREE*
- *CHECK*
- *DEV_NAME*
- *DISPLAY_WIDTH*
- *DJ_OPEN*
- *DJ_OPEN_DIR*
- *DJ_OPEN_IN*
- *DJ_OPEN_NEW*
- *DJ_OPEN_OVER*
- *DJTK_VER$*
- *FETCH_BYTES*
- *FILE_BACKUP*
- *FILE_DATASPACE*
- *FILE_LENGTH*
- *FILE_POSITION*
- *FILE_TYPE*
- *FILE_UPDATE*
- *FILLMEM_B*
- *FILLMEM_L*
- *FILLMEM_W*
- *FLUSH_CHANNEL*
- *GET_BYTE*
- *GET_FLOAT*
- *GET_LONG*
- *GET_STRING*
- *GET_WORD*
- *KBYTES_FREE*
- *LEVEL2*
- *MAX_CON*
- *MAX_DEVS*
- *MOVE_MEM*

- *MOVE_POSITION*
- *PEEK_FLOAT*
- *PEEK_STRING*
- *POKE_FLOAT*
- *POKE_STRING*
- *PUT_BYTE*
- *PUT_FLOAT*
- *PUT_LONG*
- *PUT_STRING*
- *PUT_WORD*
- *QPTR*
- *READ_HEADER*
- *RELEASE_HEAP*
- *RESERVE_HEAP*
- *SCREEN_BASE*
- *SCREEN_MODE*
- *SEARCH_C*
- *SEARCH_I*
- *SET_HEADER*
- *SET_XINC*
- *SET_YINC*
- *SYSTEM_VARIABLES*
- *USE_FONT*
- *WHERE_FONTS*

## 7.19 Disk Interfaces

The command in this toolkit is:

- *FLP_STEP*

## 7.20 ETAT

The command in this toolkit is:

- *ETAT*

## 7.21 Ecran Manager

The commands in this toolkit are:

- *SAUTO*
- *SCROF*
- *SCRON*
- *SLOAD*
- *SMOVE*
- *SSAVE*
- *SSHOW*
- *SSTAT*

## 7.22 Environment Variables

The commands in this toolkit are:

- *ENV_DEL*
- *ENV_LIST*

## 7.23 FACT

The command in this toolkit is:

- *FACT*

## 7.24 FKEY

The command in this toolkit is:

- *KEY*

## 7.25 FN

The commands in this toolkit are:

- *FNAME$*
- *KEYW*
- *PINF$*
- *QDOS$*
- *QFLIM*
- *QuATARI*
- *RMODE*
- *SCREEN*
- *SCRINC*

- *SYS_BASE*
- *THING*
- *TH_VER$*
- *WIN_BASE*
- *WINF$*

### 7.25.1 FN v1.02 Onwards

This toolkit adds one extra command to then list in the FN toolkit above. This is:

- *DEFAULT_SCR*

## 7.26 FONTS

The command in this toolkit is:

- *S_FONT*

## 7.27 FRACT

The command in this toolkit is:

- *FRACT*

## 7.28 Fast PLOT/DRAW Toolkit

The commands in this toolkit are:

- *DRAW*
- *PLOT*
- *REFRESH*
- *SCLR*
- *SCRBASE*

## 7.29 GETSTUFF

The command in this toolkit is:

- *GET_STUFF$*

## 7.30 Gold Card

The commands provided by the Gold Card ROM are:

- *CACHE_OFF*
- *CACHE_ON*
- *DEV_LIST*

- *DEV_NEXT*
- *DEV_USE*
- *DEV_USE$*
- *FLP_DENSITY*
- *FLP_EXT*
- *FLP_JIGGLE*
- *FLP_SEC*
- *FLP_START*
- *FLP_STEP*
- *FLP_TRACK*
- *FLP_USE*
- *PAR_USE*
- *PROT_DATE*
- *PRT_ABT*
- *PRT_USE*
- *RAM_USE*
- *RES_128*
- *RES_SIZE*
- *SCR2DIS*
- *SCR2EN*
- *SDP_DEV*
- *SDP_KEY*
- *SDP_SET*
- *SDUMP*
- *SER_PAUSE*
- *WIN2*

### 7.30.1 Gold Card - v2.24

The additional commands provided by this ROM are:

- *SLUG*

### 7.30.2 Gold Card - v2.67

The additional commands provided by this ROM are:

- *AUTO_DIS*
- *AUTO_TK2F1*

- *AUTO_TK2F2*

## 7.31 GPOINT

The commands in this toolkit are:

- *GPOINT*
- *POINT*

## 7.32 HCO

The commands in this toolkit are:

- *BICOP*
- *BLOOK*
- *BMOVE*
- *COL*
- *DOTLIN*
- *GETXY*
- *INVXY*
- *LDRAW*
- *PAINT*
- *SET*
- *XDRAW*

## 7.33 HOTKEY II

The commands in this toolkit are:

- *ERT*
- *EXEP*
- *HOT_CHP1*
- *HOT_CHP*
- *HOT_CMD*
- *HOT_DO*
- *HOT_GO*
- *HOT_KEY*
- *HOT_LIST*
- *HOT_LOAD1*
- *HOT_LOAD*
- *HOT_NAME$*

- *HOT_OFF*
- *HOT_PICK*
- *HOT_REMV*
- *HOT_RES1*
- *HOT_RES*
- *HOT_SET*
- *HOT_STOP*
- *HOT_STUFF*
- *HOT_THING*
- *HOT_TYPE*
- *HOT_WAKE*

## 7.34 Hard Disk Driver

The command provided in the hard disk driver is:

- *WIN_USE*

## 7.35 History Device

The commands in this toolkit are:

- *HIS_SET*
- *HIS_SIZE*
- *HIS_UNSET*
- *HIS_USE$*
- *HIS_USE*

## 7.36 Hyper

The commands in this toolkit are:

- *ARCOSH*
- *ARCOTH*
- *ARSINH*
- *ARTANH*
- *COSH*
- *COTH*
- *SINH*
- *TANH*

## 7.37 Hyperbola

The commands in this toolkit are:

- *COSH*
- *SINH*
- *TANH*

## 7.38 KEYMAN

The commands in this toolkit are:

- *KEY*
- *NOKEY*

## 7.39 KILL

The command in this toolkit is:

- *KILL*

## 7.40 LWCUPC

The commands in this toolkit are:

- *LWC$*
- *UPC$*

## 7.41 Level-2 Device Drivers

The commands in the Level 2 and/or Level 3 drivers are:

- *FBKDT*
- *FMAKE_DIR*
- *FVERS*
- *MAKE_DIR*
- *SET_FBKDT*
- *SET_FUPDT*
- *SET_FVERS*

## 7.42 MINMAX2

The commands in this toolkit are:

- *MAX*
- *MIN*

## 7.43 MULTI

The commands in this toolkit are:

- *IS_BASIC*
- *P_ENV*

## 7.44 Math Package

The commands in this toolkit are:

- *ATN*
- *ATN2*
- *BINOM*
- *CEIL*
- *DET*
- *DIV*
- *EASTER*
- *EPS*
- *FACT*
- *GCD*
- *GREGOR*
- *INF*
- *INTMAX*
- *LCM*
- *LOG2*
- *MATADD*
- *MATCOUNT*
- *MATCOUNT1*
- *MATDEV*
- *MATEQU*
- *MATIDN*
- *MATINPUT*
- *MATINV*
- *MATMAX*
- *MATMEAN*
- *MATMIN*
- *MATMULT*

- *MATPLOT*
- *MATPLOT_R*
- *MATPROD*
- *MATREAD*
- *MATRND*
- *MATSEQ*
- *MATSUB*
- *MATSUM*
- *MATTRN*
- *MAX*
- *MIN*
- *MOD*
- *NDIM*
- *SGN*
- *SIZE*
- *SQR*
- *SWAP*

## 7.45 Minerva

The commands in the Minerva ROM, over and above the standard QL ROM are:

- *CMD$*
- *MB*
- *WINDOW*

## 7.46 Minerva - Trace Toolkit

The commands in this toolkit are:

- *SSTEP*
- *TROFF*
- *TRON*

## 7.47 Minerva Extensions Toolkit

The command in this toolkit is:

- *I2C_IO*

## 7.48 NDIM

The command in this toolkit is:

- *NDIM%*

## 7.49 PAR/SER Interfaces

The command in the PAR/SER Interface ROM is:

- *PAR_USE*

## 7.50 PEX

The commands in this toolkit are:

- *IS_PEON*
- *IS_PTRAP*
- *MODE* |
- *OUTL*
- *PEOFF*
- *PEON*
- *PEX$*
- *PEX_INI*
- *PEX_SAVE*
- *PEX_XTD*
- *PICK%*
- *PIF$*
- *PX1ST*
- *PXOFF*
- *PXON*
- *QL_PEX*

### 7.50.1 PEX - v20

This version of the PEX toolkit provides an additional command which is:

- *WMOV*

## 7.51 PICEXT

The commands in this toolkit are:

- *LOADPIC*
- *SAVEPIC*

## 7.52 PIE

The commands in this toolkit are:

- *PIE_EX_OFF*
- *PIE_EX_ON*
- *PIE_OFF*
- *PIE_ON*

## 7.53 PRIO

The command in this toolkit is:

- *PRIO*

## 7.54 PTRRTP

The commands in this toolkit are:

- *PTR_X*
- *PTR_Y*
- *RTP_R*
- *RTP_T*

## 7.55 Path device

The commands provided by the Path device are:

- *PTH_ADD*
- *PTH$*
- *PTH_LIST*
- *PTH_RMV*
- *PTH_USE*
- *PTH_USE$*

## 7.56 Pointer Interface - v1.23 Onwards

The commands in this toolkit are:

- *CKEYOFF*
- *CKEYON*

## 7.57 QL ROM

The commands in QL ROMs prior to version JM are:

- *ABS*
- *ACOS*
- *ACOT*
- *ADATE*
- *AND*
- *ARC*
- *ARC_R*
- *ASIN*
- *AT*
- *ATAN*
- *BAUD*
- *BEEP*
- *BEEPING*
- *BLOCK*
- *BORDER*
- *CALL*
- *CHR$*
- *CIRCLE*
- *CIRCLE_R*
- *CLEAR*
- *CLOSE*
- *CLS*
- *CODE*
- *CONTINUE*
- *COPY*
- *COPY_N*
- *COS*
- *COT*
- *CSIZE*
- *CURSOR*
- *DATA*
- *DATE*

- *DATE$*
- *DAY$*
- *DEFine FuNction*
- *DEFine PROCedure*
- *DEFine xxx*
- *DEG*
- *DELETE*
- *DIM*
- *DIMN*
- *DIR*
- *DIV*
- *DLINE*
- *EDIT*
- *ELLIPSE*
- *ELLIPSE_R*
- *ELSE*
- *END*
- *END DEFine*
- *END FOR*
- *END IF*
- *END REPeat*
- *END SELect*
- *EOF*
- *ERR_XX*
- *EXEC*
- *EXEC_W*
- *EXIT*
- *EXP*
- *FILL*
- *FILL$*
- *FLASH*
- *FOR*
- *FORMAT*
- *FuNction*

- *GO SUB*
- *GO TO*
- *IF*
- *INK*
- *INKEY$*
- *INPUT*
- *INSTR*
- *INT*
- *KEYROW*
- *LBYTES*
- *LEN*
- *LET*
- *LINE*
- *LINE_R*
- *LIST*
- *LN*
- *LOAD*
- *LOCal*
- *LOG10*
- *LRUN*
- *MERGE*
- *MISTake*
- *MOD*
- *MODE*
- *MOVE*
- *MRUN*
- *NET*
- *NEW*
- *NEXT*
- *NOT*
- *ON…GO SUB*
- *OPEN*
- *OPEN_IN*
- *OPEN_NEW*

- *OR*
- *OVER*
- *PAN*
- *PAPER*
- *PAUSE*
- *PEEK_L*
- *PENDOWN*
- *PENUP*
- *PI*
- *POINT*
- *POINT_R*
- *POKE_L*
- *PRINT*
- *PROCedure*
- *RAD*
- *RANDOMISE*
- *READ*
- *RECOL*
- *REMAINDER*
- *REMark*
- *RENUM*
- *REPeat*
- *RESPR*
- *RESTORE*
- *RETRY*
- *RETurn*
- *RND*
- *RUN*
- *SAVE*
- *SBYTES*
- *SCALE*
- *SCROLL*
- *SDATE*
- *SELect*

- *SELect ON*
- *SEXEC*
- *SIN*
- *SQRT*
- *STEP*
- *STOP*
- *STRIP*
- *SUB*
- *TAN*
- *THEN*
- *TO*
- *TURN*
- *TURNTO*
- *UNDER*
- *VER$*
- *WIDTH*
- *WINDOW*
- *XOR*

### 7.57.1 QL ROM JM Onwards

The JM ROM provided the following additional commands:

- *END WHEN*
- *ERLIN*
- *ERNUM*
- *ERRor*
- *REPORT*
- *TRA*
- *WHEN condition*
- *WHEN ERRor*

## 7.58 QPC / QXL

The commands in the QXL ROM and QPC are:

- *WIN_DRIVE*
- *WIN_DRIVE$*
- *WIN_FORMAT*

- *WIN_REMV*
- *WIN_START*
- *WIN_STOP*
- *WIN_USE*
- *WIN_WP*

## 7.59 QSOUND

The commands in this toolkit are:

- *BELL*
- *CHANNELS*
- *CURDIS*
- *CURSEN*
- *EXPLODE*
- *EXTRAS*
- *LEFT*
- *PLAY*
- *RELEASE*
- *SHOOT*

## 7.60 QView Tiny Toolkit

The Qview Tiny Toolkit, not to be confused with *TinyToolkit*, provides the following commands:

- *TTALL*
- *TT$*
- *TTEDELETE*
- *TTEFP*
- *TTEOPEN*
- *TTET3*
- *TTEX*
- *TTEX_W*
- *TTFINDM*
- *TTINC*
- *TTME%*
- *TTMODE%*
- *TTPEEK$*
- *TTPOKE$*

- *TTPOKEM*
- *TTREL*
- *TTRENAME*
- *TTSUS*
- *TTV*

## 7.61 QVME - Level E-19 Drivers onwards

The commands in this toolkit are:

- *DISP_BLANK*
- *DISP_RATE*
- *DISP_SIZE*

## 7.62 QXL

The commands supplied in the QXL ROM are:

- *DISP_UPDATE*
- *PRT_ABT*
- *PRT_USE*
- *WIN_DRIVE*
- *WIN_DRIVE$*
- *WIN_FORMAT*
- *WIN_REMV*
- *WIN_START*
- *WIN_STOP*
- *WIN_USE*
- *WIN_WP*

## 7.63 Qjump RAMPRT

The command in this toolkit is:

- *PRT_USE*

## 7.64 RES

The command in this toolkit is:

- *RESET*

## 7.65 REV

The command in this toolkit is:

- *REV$*

## 7.66 SDUMP_REXT

## 7.67 SERMouse

The commands in this toolkit are:

- *BAUDRATE*
- *BLS*
- *SERMAWS*
- *SERMCUR*
- *SERMOFF*
- *SERMON*
- *SERMPTR*
- *SERMRESET*
- *SERMSPEED*
- *SERMWAIT*

## 7.68 SMS

The commands provided by SMS are:

- *CACHE_OFF*
- *CACHE_ON*
- *CMD$*
- *DEV_LIST*
- *DEV_NEXT*
- *DEVTYPE*
- *DEV_USE*
- *DEV_USE$*
- *EOFW*
- *EPROM_LOAD*
- *FBKDT*
- *FLP_DENSITY*
- *FLP_SEC*
- *FLP_START*

- *FLP_STEP*
- *FLP_TRACK*
- *FLP_USE*
- *FSERVE*
- *IO_PRIORITY*
- *JOB_NAME*
- *LANGUAGE*
- *LANGUAGE$*
- *LANG_USE*
- *PEEK$*
- *POKE$*
- *PROT_DATE*
- *QLOAD*
- *QLRUN*
- *QMERGE*
- *QMRUN*
- *QSAVE*
- *QSAVE_O*
- *QUIT*
- *RAM_USE*
- *SBASIC*
- *SLUG*

### 7.68.1 SMS - v2.31

The additional command in this version of SMS is:

- *KBD_TABLE*

## 7.69 SMSQ

The commands in this manual for SMSQ are:

- *SB_THING*

### 7.69.1 SMSQ - 3.26

An additional command in SMSQ 3.26 onwards is:

- *ALLOCATION*

## 7.70 SMSQ/E

SMSQ/E provides the following commands:

- *CD_ALLTIME*
- *CD_CLOSE*
- *CD_EJECT*
- *CD_FIRSTTRACK*
- *CD_HOUR*
- *CD_HSG2RED*
- *CD_INIT*
- *CD_ISCLOSED*
- *CD_ISINSERTED*
- *CD_ISPAUSED*
- *CD_ISPLAYING*
- *CD_LASTTRACK*
- *CD_LENGTH*
- *CD_MINUTE*
- *CD_PLAY*
- *CD_RED2HSG*
- *CD_RESUME*
- *CD_SECOND*
- *CD_STOP*
- *CD_TRACK*
- *CD_TRACKLENGTH*
- *CD_TRACKSTART*
- *CD_TRACKTIME*
- *CHK_HEAP*
- *DAY%*
- *DEV_USEN*
- *DISP_INVERSE*
- *DISP_SIZE*
- *DISP_TYPE*
- *DISP_UPDATE*
- *DOS_DRIVE*
- *DOS_DRIVE$*

- *DOS_USE*
- *FET*
- *FEW*
- *FEX*
- *FEX_M*
- *FLP_DENSITY*
- *FLP_DRIVE*
- *FLP_DRIVE$*
- *FLP_SEC*
- *FLP_STEP*
- *FLP_USE*
- *HGET*
- *HOT_GETSTUFF$*
- *HPUT*
- *JOBID*
- *LGET*
- *LPUT*
- *MACHINE*
- *MIDINET*
- *MONTH%*
- *MNET*
- *MNET_OFF*
- *MNET_ON*
- *MNET%*
- *MNET_S%*
- *MNET_USE*
- *MOUSE_SPEED*
- *MOUSE_STUFF*
- *OUTLN*
- *PAR_ABORT*
- *PAR_BUFF*
- *PAR_CLEAR*
- *PAR_DEFAULTPRINTER$*
- *PAR_GETFILTER*

- *PAR_GETPRINTER$*
- *PAR_PRINTERCOUNT*
- *PAR_PRINTERNAME$*
- *PAR_PULSE*
- *PAR_SETFILTER*
- *PAR_SETPRINTER*
- *PAR_USE*
- *PEEKS_L*
- *POKES_L*
- *PROCESSOR*
- *PROT_MEM*
- *PRT_ABORT*
- *PRT_BUFF*
- *PRT_CLEAR*
- *PRT_USE*
- *PRT_USE$*
- *QPC_CMDLINE$*
- *QPC_EXEC*
- *QPC_EXIT*
- *QPC_HOSTOS*
- *QPC_MAXIMIZE*
- *QPC_MINIMIZE*
- *QPC_MSPEED*
- *QPC_NETNAME$*
- *QPC_QLSCREMU*
- *QPC_RESTORE*
- *QPC_SYNCSCRAP*
- *QPC_VER$*
- *QPC_WINDOWSIZE*
- *QPC_WINDOWTITLE*
- *RESET*
- *SB_THING*
- *SCR_BASE*
- *SCR_LLEN*

- *SCR_XLIM*
- *SCR_YLIM*
- *SER_ABORT*
- *SER_BUFF*
- *SER_CDEOF*
- *SER_CLEAR*
- *SER_FLOW*
- *SER_GETPORT$*
- *SERNET*
- *SER_PAUSE*
- *SER_ROOM*
- *SER_SETPORT*
- *SER_USE*
- *SNET*
- *SNET%*
- *SNET_ROPEN*
- *SNET_S%*
- *SNET_USE*
- *TH_FIX*
- *WEEKDAY%*
- *WGET*
- *WHEN condition*
- *WIN_DRIVE*
- *WIN_DRIVE$*
- *WIN_REMV*
- *WIN_SLUG*
- *WIN_START*
- *WIN_STOP*
- *WIN_USE*
- *WIN_WP*
- *WPUT*
- *YEAR%*

### 7.70.1 SMSQ/E - v2.50 Onwards

This version provided the following additional command(s):

- *HOT_THING1*

### 7.70.2 SMSQ/E - v2.55 Onwards

This version provided the following additional command(s):

- *UPUT*

### 7.70.3 SMSQ/E - v2.58 Onwards

This version provided the following additional command(s):

- *INSTR_CASE*

### 7.70.4 SMSQ/E - v2.71 Onwards

This version provided the following additional command(s):

- *SEND_EVENT*
- *WAIT_EVENT*

### 7.70.5 SMSQ/E - v2.73 Onwards

This version provided the following additional command(s):

- *DMEDIUM_DENSITY*
- *DMEDIUM_DRIVE$*
- *DMEDIUM_FORMAT*
- *DMEDIUM_FREE*
- *DMEDIUM_NAME$*
- *DMEDIUM_RDONLY*
- *DMEDIUM_REMOVE*
- *DMEDIUM_TOTAL*
- *DMEDIUM_TYPE*

### 7.70.6 SMSQ/E - v2.98 Onwards

This version provided the following additional command(s):

- *BGCOLOUR_24*
- *BGCOLOUR_QL*
- *BGIMAGE*
- *COLOUR_24*
- *COLOUR_NATIVE*
- *COLOUR_PAL*

- *COLOUR_QL*
- *PALETTE_8*
- *PALETTE_QL*

### 7.70.7 SMSQ/E - v3.00 Onwards

This version provided the following additional command(s):

- *WM_BLOCK*
- *WM_BORDER*
- *WM_INK*
- *WM_PAPER*
- *WM_STRIP*

### 7.70.8 SMSQ/E - v3.01 Onwards

This version provided the following additional command(s):

- *WM_MOVEMODE*

### 7.70.9 SMSQ/E - v3.12 Onwards

This version provided the following additional command(s):

- *PE_BGOFF*
- *PE_BGON*

### 7.70.10 SMSQ/E - v2.73 for Atari

Additional command(s) in this version of SMSQ/E are:

- *WIN_FORMAT*

SMSQ/E for Atari Additional command(s) in this version of SMSQ/E are:

- *PAR_PULSE*
- *WIN_DRIVE*
- *WIN_DRIVE$*
- *WIN_REMV*
- *WIN_SLUG*
- *WIN_START*
- *WIN_STOP*
- *WIN_USE*
- *WIN_WP*

SMSQ/E for Atari ST & TT Additional command(s) in this version of SMSQ/E are:

- *DISP_INVERSE*

### 7.70.11 SMSQ/E for Q68

The commands in this toolkit are:

- *ALFM*
- *CARD_CREATE*
- *CARD_DIR$*
- *CARD_INIT*
- *CARD_RENF*
- *DISP_MODE*
- *FAT_DRIVE*
- *FAT_DRIVE$*
- *FAT_USE*
- *FAT_WP*
- *FREE_FMEM*
- *KILLSOUND*
- *QUB_DRIVE*
- *QUB_DRIVE$*
- *QUB_USE*
- *QUB_WP*
- *SLUG*
- *SOUNDFILE*
- *SOUNDFILE2*
- *SOUNDFILE3*
- *WIN_CHECK*
- *WIN_DRIVE*
- *WIN_DRIVE$*
- *WIN_FORMAT*
- *WIN_SAFE*
- *WIN_USE*
- *WIN_WP*

### 7.70.12 SMSQ/E for QPC

The commands in this toolkit are:

- *CD_ALLTIME*
- *CD_CLOSE*
- *CD_EJECT*

- *CD_FIRSTTRACK*
- *CD_HOUR*
- *CD_HSG2RED*
- *CD_INIT*
- *CD_ISCLOSED*
- *CD_ISINSERTED*
- *CD_ISPAUSED*
- *CD_ISPLAYING*
- *CD_LASTTRACK*
- *CD_LENGTH*
- *CD_MINUTE*
- *CD_PLAY*
- *CD_RED2HSG*
- *CD_RESUME*
- *CD_SECOND*
- *CD_STOP*
- *CD_TRACK*
- *CD_TRACKLENGTH*
- *CD_TRACKSTART*
- *CD_TRACKTIME*
- *DOS_DRIVE*
- *DOS_DRIVE$*
- *DOS_USE*
- *FLP_DENSITY*
- *FLP_DRIVE*
- *FLP_DRIVE$*
- *FLP_SEC*
- *FLP_STEP*
- *FLP_USE*
- *MACHINE*
- *MOUSE_SPEED*
- *MOUSE_STUFF*
- *PAR_DEFAULTPRINTER$*
- *PAR_GETFILTER*

- *PAR_GETPRINTER$*
- *PAR_PRINTERCOUNT*
- *PAR_PRINTERNAME$*
- *PAR_SETFILTER*
- *PAR_SETPRINTER*
- *QPC_CMDLINE$*
- *QPC_EXEC*
- *QPC_EXIT*
- *QPC_HOSTOS*
- *QPC_MAXIMIZE*
- *QPC_MINIMIZE*
- *QPC_MSPEED*
- *QPC_NETNAME$*
- *QPC_QLSCREMU*
- *QPC_RESTORE*
- *QPC_SYNCSCRAP*
- *QPC_VER$*
- *QPC_WINDOWSIZE*
- *QPC_WINDOWTITLE*
- *SER_GETPORT$*
- *SER_SETPORT*

## 7.71 ST/QL

The commands in this toolkit are:

- *ACCEL_OFF*
- *ACCEL_ON*
- *ACCEL_SET*
- *ACCEL_STATE*
- *APPEND*
- *BELL*
- *DEV_LIST*
- *DEV_NEXT*
- *DEV_USE*
- *DEV_USE$*

- *EXPLODE*
- *FLP_TRACK*
- *FLP_USE*
- *GER_MSG*
- *GER_TRA*
- *NOR_MSG*
- *NOR_TRA*
- *PAR_ABORT*
- *PAR_BUFF*
- *PAR_CLEAR*
- *PAR_PULSE*
- *PAR_USE*
- *PLAY*
- *PRT_ABORT*
- *PRT_BUFF*
- *PRT_CLEAR*
- *PRT_USE*
- *PRT_USE$*
- *RAM_USE*
- *RELEASE*
- *SDP_DEV*
- *SDP_KEY*
- *SDP_SET*
- *SDUMP*
- *SER_ABORT*
- *SER_BUFF*
- *SER_CLEAR*
- *SER_FLOW*
- *SER_ROOM*
- *SER_USE*
- *SHOOT*
- *WIN_DRIVE*
- *WIN_SLUG*
- *WIN_START*

- *WIN_STOP*
- *WIN_USE*

### 7.71.1 ST/QL - Pre v2.24

This toolkit provides the following, additional, command:

- *A_BLANK*

### 7.71.2 ST/QL - Level B-11 Onwards

This toolkit provides the following, additional, command:

- *TH_FIX*

### 7.71.3 ST/QL - Level C-17 Onwards

This toolkit provides the following, additional, command:

- *KBD_TABLE*

### 7.71.4 ST/QL - Level C-19 Onwards

This toolkit provides the following, additional, command:

- *WIN2*

### 7.71.5 ST/QL - Level C-20 Onwards

This toolkit provides the following, additional, command:

- *WIN_REMV*

### 7.71.6 ST/QL - Level D00 Onwards

This toolkit provides the following, additional, command:

- *SER_CDEOF*

### 7.71.7 ST/QL - level D.02 Onwards

This toolkit provides the following, additional, command:

- *FLP_START*

## 7.72 STAMP

The command in this toolkit is:

- *STAMP*

## 7.73 SWAP

The commands in this toolkit are:

- *SWAP*
- *W_SWAP*

## 7.74 SYSBASE

The command in this toolkit is:

- *SYS_BASE*

## 7.75 Shape Toolkit

The commands in this toolkit are:

- *ALINE*
- *APOINT*
- *DEMO*

## 7.76 Super Gold Card

The command in the SGC ROM is:

- *SLUG*

## 7.77 SuperQBoard

The command in this toolkit is:

- *PAR_USE*

## 7.78 SuperWindow Toolkit

The commands in this toolkit are:

- *SCR_REFRESH*
- *SCR_SIZE*
- *SCR_STORE*

## 7.79 THOR

Commands provided on THOR machines are:

- *CLOSE*
- *FLP_SEC*
- *FLP_START*
- *FLP_TRACK*
- *FLP_USE*
- *LANGUAGE$*
- *SET_CLOCK*
- *SET_LANGUAGE*
- *TOP_WINDOW*

- *WCOPY*
- *WCOPY_F*
- *WCOPY_O*
- *WDEL*
- *WDEL_F*
- *WDIR*
- *WSTAT*

### 7.79.1 THOR 8

The THOR 8 added an additional command:

- *WMON*

### 7.79.2 THOR 8 - v4.20 Onwards

The additional command(s) in the version are:

- *WTV*

### 7.79.3 THOR XVI

Commands provided on THOR XVI machines are:

- *ALCHP*
- *BGET*
- *BIN*
- *BIN$*
- *BPUT*
- *CDEC$*
- *CHAR_INC*
- *CHAR_USE*
- *CLCHP*
- *CLOCK*
- *COPY*
- *COPY_N*
- *COPY_O*
- *CURDIS*
- *CURSEN*
- *DATA_USE*
- *EW*
- *EX*

- *EXTRAS*
- *FDAT*
- *FDEC$*
- *FLEN*
- *FOP_DIR*
- *FOPEN*
- *FOP_IN*
- *FOP_NEW*
- *FOP_OVER*
- *FPOS*
- *FREE_MEM*
- *FSERVE*
- *FTYP*
- *GET*
- *HEX*
- *HEX$*
- *IDEC$*
- *IO_TRAP*
- *JOBS*
- *LRESPR*
- *MAKE_DIR*
- *NET_ID*
- *NFS_USE*
- *NO_CLOCK*
- *OPEN_DIR*
- *OPEN_OVER*
- *PARTYP*
- *PARUSE*
- *PROG_USE*
- *PUT*
- *RECHP*
- *RENAME*
- *REPORT*
- *RJOB*

- *SAVE_O*
- *SBYTES_O*
- *SEXEC_O*
- *SPJOB*
- *SPL*
- *SPLF*
- *SPL_USE*
- *STAT*
- *SYS_VARS*
- *TRUNCATE*
- *VIEW*
- *WHEN condition*
- *WHEN ERRor*
- *WIN2*
- *WINDOW*
- *WIN_USE*
- *WMON*
- *WTV*

## 7.80 TRIM

The command in this toolkit is:

- *TRIM$*

## 7.81 TRIPRODRO

The commands in this toolkit are:

- *DROUND*
- *PROUND*
- *TRINT*

## 7.82 TRUFA

The commands in this toolkit are:

- *FALSE%*
- *TRUE%*

## 7.83 TinyToolkit

TinyToolkit, not to be confused with *QView Tiny Toolkit*, provides the following commands:

- *BASIC_L*
- *BASICP*
- *BCLEAR*
- *BREAK_OFF*
- *CBASE*
- *CHANGE*
- *CHANID*
- *CHANNELS*
- *CLEAR_HOT*
- *CLOSE%*
- *CLRMDV*
- *CUR*
- *DEVLIST*
- *ELIS*
- *FILE_DAT*
- *FILE_LEN*
- *FILE_POS*
- *FILE_PTRA*
- *FILE_PTRR*
- *FLIS*
- *FORCE_TYPE*
- *FREAD*
- *FWRITE*
- *GET_BYTE$*
- *GRAB*
- *HEADR*
- *HEADS*
- *HOT*
- *JBASE*
- *KJOB*
- *KJOBS*
- *NEW_NAME*

- *ODD*
- *PEEK$*
- *PEND*
- *POKE$*
- *QDOS$*
- *QRAM$*
- *RAND*
- *RELEASE*
- *REL_JOB*
- *REPORT*
- *RESET*
- *ROM*
- *SEARCH*
- *SJOB*
- *S_LOAD*
- *S_SAVE*
- *S_SHOW*
- *SXTRAS*
- *TCONNECT*
- *TINY_EXT*
- *TINY_RMV*
- *TXTRAS*
- *TYPE*
- *UPPER$*
- *WBASE*
- *WMAN$*
- *ZAP*

### 7.83.1 TinyToolkit - Pre v1.10

The commands in versions of TinyToolkit prior to 1.10 is:

- *SPJOB*

### 7.83.2 TinyToolkit - v1.10 Onwards

The additional command provided in these versions is:

- *SP_JOB*

## 7.84 Toolfin

The commands in this toolkit are:

- *MT*
- *RAE*
- *RAFE*
- *TCA*
- *TEE*
- *TNC*
- *VAR*
- *VAR*
- *VFR*

## 7.85 Toolkit II

The commands in this toolkit are:

- *AJOB*
- *ALARM*
- *ALCHP*
- *ALTKEY*
- *BGET*
- *BIN*
- *BIN$*
- *BPUT*
- *CDEC$*
- *CHAR_INC*
- *CHAR_USE*
- *CLCHP*
- *CLOCK*
- *CLOSE*
- *CONTINUE*
- *COPY*
- *COPY_H*
- *COPY_N*
- *COPY_O*
- *CURDIS*

- *CURSEN*
- *DATAD$*
- *DATA_USE*
- *DDOWN*
- *DEL_DEFB*
- *DELETE*
- *DESTD$*
- *DEST_USE*
- *DIR*
- *DLIST*
- *DNEXT*
- *DO*
- *DUP*
- *ED*
- *ET*
- *EW*
- *EX*
- *EXEC*
- *EXEC_W*
- *EXTRAS*
- *FDAT*
- *FDEC$*
- *FEXP$*
- *FLEN*
- *FLUSH*
- *FNAME$*
- *FOP_DIR*
- *FOPEN*
- *FOP_IN*
- *FOP_NEW*
- *FOP_OVER*
- *FPOS*
- *FREE_MEM*
- *FTEST*

- *FTYP*
- *FUPDT*
- *FXTRA*
- *GET*
- *HEX*
- *HEX$*
- *IDEC$*
- *JOB$*
- *JOBS*
- *LBYTES*
- *LOAD*
- *LRESPR*
- *LRUN*
- *MERGE*
- *MRUN*
- *NEW*
- *NFS_USE*
- *NXJOB*
- *OJOB*
- *OPEN*
- *OPEN_DIR*
- *OPEN_IN*
- *OPEN_NEW*
- *OPEN_OVER*
- *PARNAM$*
- *PARSTR$*
- *PARTYP*
- *PARUSE*
- *PJOB*
- *PRINT_USING*
- *PROGD$*
- *PROG_USE*
- *PUT*
- *RECHP*

- *RENAME*
- *REPORT*
- *RETRY*
- *RJOB*
- *SAVE*
- *SAVE_O*
- *SBYTES*
- *SBYTES_O*
- *SEXEC*
- *SEXEC_O*
- *SPJOB*
- *SPL*
- *SPLF*
- *SPL_USE*
- *STAT*
- *STOP*
- *TK2_EXT*
- *TRUNCATE*
- *VIEW*
- *WCOPY*
- *WDEL*
- *WDIR*
- *WMON*
- *WREN*
- *WSTAT*
- *WTV*

### 7.85.1 Toolkit II - Hardware Version Only or SMS

The hardware (ROM, Disc INterface etc) and SMS versions of Toolkit II provide the following command:

- *FSERVE*

## 7.86 Trump Card

The commands supplied in the Trump Card ROM are:

- *FLP_SEC*
- *FLP_START*

- *FLP_TRACK*
- *FLP_USE*
- *PRT_ABT*
- *PRT_USE*
- *RAM_USE*
- *RES_128*
- *SDP_DEV*
- *SDP_KEY*
- *SDP_SET*
- *SDUMP*

## 7.87 Turbo Toolkit

The commands in this toolkit are:

- *ALLOCATION*
- *BASIC_F*
- *BASIC_INDEX%*
- *BASIC_L*
- *BASIC_NAME$*
- *BASIC_POINTER*
- *BASIC_TYPE%*
- *CATNAP*
- *CHANNEL_ID*
- *CHARGE*
- *COMMAND_LINE*
- *COMPILED*
- *CONNECT*
- *CURSOR_OFF*
- *CURSOR_ON*
- *DATA_AREA*
- *DATASPACE*
- *DEALLOCATE*
- *DEFAULT_DEVICE*
- *DEVICE_SPACE*
- *DEVICE_STATUS*

- *EDIT$*
- *EDITF*
- *EDIT%*
- *END_CMD*
- *END_WHEN*
- *ERLIN%*
- *ERNUM%*
- *TK_VER$*

### 7.87.1 Turbo Toolkit - v3.00

This version of the toolkit added the following commands:

- *TURBO_diags*
- *TURBO_F*
- *TURBO_locstr*
- *TURBO_model*
- *TURBO_objdat*
- *TURBO_objfil*
- *TURBO_optim*
- *TURBO_P*
- *TURBO_repfil*
- *TURBO_struct*
- *TURBO_taskn*
- *TURBO_window*

### 7.87.2 Turbo Toolkit - v3.20

This version of the toolkit added the following commands:

- *DEBUG*
- *DEBUG_LEVEL*

## 7.88 UNJOB

The command in this toolkit is:

- *UNJOB*

## 7.89 WIPE

The command in this toolkit is:

- *WIPE*

## 7.90 WM

The command in this toolkit is:

- *WM*

## 7.91 XKBD

The command in this toolkit is:

- *KBD_USE*

CHAPTER

EIGHT

# KEYWORDS A

## 8.1 ABS

| | |
|---|---|
| Syntax | ABS (number) or<br>ABS (number1 *[,number^x]*) (Minerva only) |
| Location | QL ROM |

This function returns the absolute value of a number - ie. the positive difference (or distance) between zero and the number. The absolute value of a positive number (including zero) therefore, is the number itself - negative numbers are converted to positive. This function will happily handle 32-bit integer numbers (-INTMAX..INTMAX, roughly -1E9..1E9).

**Example 1**

The SIGN% function returns 1 if the supplied parameter is positive, -1 if negative, or 0 if it is zero, for example,

PRINT SIGN%(-10) will print -1 on screen.

This version rounds values which are very close to zero (use = in line 110 instead of == if you want to avoid this).

Note that line 110 is needed to avoid an error when line 120 tries to divide by zero.

```
100 DEFine FuNction SIGN% (number)
110  IF number==0 THEN RETurn 0
120  RETurn number/ABS(number)
130 END DEFine
```

**Example 2**

Here is a simple implementation of the cosine function. Of course, it cannot compete with the speed of a machine code function, but it allows you to specify the precision of the result. You can optimise the function by exploiting the symmetries of the cosine function.

```
100 DEFine FuNction MYCOS (x, prec)
110  LOCal fct, result, xpower, i, lagrange, sqrx
120  fct = 1: result = 1
130  xpower = 1: sqrx = x*x
140  i = 2
150  REPeat taylor
```

(continues on next page)

(continued from previous page)

```
160    fct = fct * (i-1) * i
170    xpower = - xpower * sqrx
180    result = result + xpower/fct
190    lagrange = ABS(xpower*x / fct / (i+1))
200    IF lagrange < prec THEN EXIT taylor
210    i = i + 2
220  END REPeat taylor
230  RETurn result
240 END DEFine MYCOS
```

**MINERVA NOTE**

ABS can accept more than one parameter. This version of ABS will square each parameter, and return the square root of the total of those squares, eg. ABS(x,y)=SQRT(x2+y2). This is therefore useful to calculate the distance between two points (using pythagoras' method).

For example, to calculate the distance between the points on screen at (10,20) and (100,75), simply type in: PRINT ABS(100-10,75-20)

Three parameters can be used to find the distance between two points in three dimensional space. Any more parameters take you into the realm of theoretical mathematics (we always thought that time was the fourth dimension!).

For example, to calculate the length of a diagonal in a standard cube (length of sides = 1), use: PRINT ABS(1,1,1)

**CROSS-REFERENCE**

See *SGN* and *SGN%* for similar machine code versions of our example function SIGN% demonstrated above.

## 8.2 ABS_POSITION

| | |
|---|---|
| Syntax | ABS_POSITION #channel, position |
| Location | DJToolkit 1.16 |

This procedure will set the file pointer to the position given for the file attached to the given channel number. If you attempt to set the position for a screen or some other non-directory device channel, you will get a bad parameter error, as you will if position is negative.

If the position given is 0, the file will be positioned to the start, if the position is a large number which is greater than the current file size, the position will be set to the end of file and no error will occur.

After an ABS_POSITION command, all file accesses will take place at the new position.

**EXAMPLE**

```
1500 REMark Set position to very end, for appending data
1510 ABS_POSITION #3, 6e6
1520 ...
```

**CROSS-REFERENCE**

*MOVE_POSITION*.

## 8.3 ACCEL_OFF

| | |
|---|---|
| Syntax | ACCEL_OFF |
| Location | ST/QL |

See ACCEL_ON below!

## 8.4 ACCEL_ON

| | |
|---|---|
| Syntax | ACCEL_ON |
| Location | ST/QL |

The ST/QL Emulator supports several of the accelerator boards which can be plugged into the Atari ST computer, thus allowing much greater operational speed. This command both enables the 16MHz mode on the Atari ST and tells the attached accelerator board to use its memory cache (if built in).

**NOTE**

This and the other ACCEL_... commands will be ignored unless you have previously used ACCEL_SET to define the type of accelerator board attached to the Atari ST.

**CROSS-REFERENCE**

*ACCEL_OFF* turns off the 16MHz mode (if possible) and also tells the accelerator board that it should no longer use its memory cache. Also see *ACCEL_SET*.

## 8.5 ACCEL_SET

| | |
|---|---|
| Syntax | ACCEL_SET type,option |
| Location | ST/QL |

Before the ST/QL Emulator can use an accelerator board plugged into the Atari ST, it is necessary to use the command ACCEL_SET to tell the Emulator about the board and to activate the board.

There are currently five accelerator boards which are recognised by the Emulator. Use the following values for type to tell the Emulator which one is attached:

- H - HyperCache (ProVME)
- A - AdSpeed (ICD)
- M - MegaSTE (ATARI)
- P - HyperCache 030 (ProVME), 68030 Board
- T - TT (ATARI)

If you have a 68030 board attached, the ST/QL Emulator can only use external caches with this board.

The option parameter currently only has any effect when HyperCache is attached. This can have the value 6 or 7 (default is 6). This is used to specify which bit of the Atari's sound chip is used to switch HyperCache. If you have the HyperCache 030 attached, you can pass the parameter 0 (default) to enable external caches only, 1 to enable the internal caches only or 2 to enable both external and internal caches.

**NOTE**

Unfortunately, due to the higher speed of the Atari ST with an accelerator board enabled, you may encounter problems with the parallel printer board - use the command PAR_PULSE.

**CROSS-REFERENCE**

See also *ACCEL_ON*, *ACCEL_OFF* and *ACCEL_STATE*.

## 8.6 ACCEL_STATE

| | |
|---|---|
| Syntax | ACCEL_STATE |
| Location | ST/QL |

This function returns the value 1 if the ST/QL Emulator has been told that an accelerator board is enabled. Otherwise, it returns the value 0.

**CROSS-REFERENCE**

*ACCEL_SET* tells the Emulator that an accelerator board is enabled.

## 8.7 ACOPY

| | |
|---|---|
| Syntax | ACOPY filename1,filename2 |
| Location | ATARIDOS |

This command is similar to COPY except that it copies a file from a QL Format disk to an Atari Format disk. No conversion takes place.

**NOTE**

You will need to pass the Atari filename in quote marks if it includes a three letter extension preceded by a dot eg:

ACOPY flp1_PROGRAM_BAS, "flp2_PROGRAM.BAS"

**CROSS-REFERENCE**

*QCOPY* copies a file from an Atari disk to a QL disk.

See *AFORMAT* and *QACONVERT*.

## 8.8 ACOS

| | |
|---|---|
| Syntax | ACOS (x) |
| Location | QL ROM |

The function ACOS, is the arc-cosine function, that is to say the opposite to the cosine function (COS in SuperBASIC). However, x must always be in the range -1…1 as the cosine of an angle can only ever be in this range. Anything outside of this range will produce an Overflow Error.

The angle returned will be in the range 0…PI with ACOS(1)=0 and ACOS(-1)=PI. This means that the maximum angle which can be found with the ACOS function is 180 degrees. It is up to you to check whether this angle appears above or below the base line of the triangle (check the co-ordinates of the corners).

Note that if a negative value of x is provided, the angle returned will be the obtuse angle (ie. greater than 90 degrees).

**Example**

To calculate the angle at which a projectile was fired which has travelled a horizontal distance of 250 metres after 10 minutes and is travelling at 3 kilometres per hour (ignoring the effects of gravity):

```
100 Speed=3:Distan=250/1000
110 Time_elapsed=10
120 Actual_distance=(Speed/60)*Time_elapsed
130 PRINT 'Projectile fired at an angle of ';
140 PRINT DEG(ACOS(Distan/Actual_distance))&' degrees'
```

**NOTE**

The angle returned will be in radians - if you wish to convert this angle to degrees, use DEG ( ACOS (x) ).

**CROSS-REFERENCE**

*COS*, *ASIN*, *SIN*, *RAD*.

Compare *ARCOSH*.

Also please see the Mathematics section in the Appendix.

## 8.9 ACOT

| | |
|---|---|
| Syntax | ACOT (x) or<br>ACOT (y,x) (Minerva v1.90+ only) |
| Location | QL ROM |

The function ACOT, is the arc-cotangent function, that is to say the inverse of the cotangent function (COT in SuperBASIC): COT(ACOT(x))=x for all values of x, but due to the periodic nature of COT, ACOT(COT(x))=x is only true for where: 0<x<PI.

Note that if a negative value of x is provided, the angle returned will be the obtuse angle (ie. greater than 90 degrees).

**MINERVA NOTE**

ACOT can accept two parameters. If you specify two parameters then ACOT(y,x) will give the angle from the origin to the point (x,y). This is actually the same as ACOT(x/y) although it does also cater for when y=0 which would otherwise give an overflow error.

**CROSS-REFERENCE**

*COT*, *ATAN*, *TAN*.

Please see the Mathematics section in the Appendix.

See also *ARCOTH*.

## 8.10 ADATE

| | |
|---|---|
| Syntax | ADATE seconds |
| Location | QL ROM |

ADATE adjusts the current system clock by the given number of seconds, so ADATE 60 would advance the internal clock by a minute and ADATE -86400 sets it back by one day.

**Example**

Apart from adjusting the clock relatively, ADATE can also be used to set the time and date absolutely. This is because the function DATE contains the system time in seconds after a fictional 'Birth Date' (Midnight on 1 January 1961 on all ROM implementations):-

ADATE -DATE will set the clock to that Birth Date (when DATE=0)

ADATE 1E9 advances the clock by roughly 31 years and nine months.

ADATEs can then be combined by adding values:

ADATE 1E9-DATE sets the clock to DATE$="1992 Sep 09 01:46:40"

**NOTE 1**

ADATE generally needs one second to execute because some ROMs (notably the THOR XVI, MG ROM and Minerva) will wait for the next full second before amending the time (therefore do not use ADATE 1 to wind the clock on!).

**NOTE 2**

Any attempts to wind the system clock back to earlier than 1st Jan 1961 will actually deduct the difference from 6th Feb 2097. However, the system clock (on implementations other than Minerva and SMS) runs into trouble here because any date later than 3.14:07 on 19th Jan 2029 should produce a negative number (!) whenever the function DATE is used. However, on non-Minerva ROMs and non-SMS systems, a positive number is produced, preventing DATE from recognising later dates.

The system clock itself, does however appear able to support dates and times between 0.0:00 on 1st Jan 1961 and 6.28:15 on 6th Feb 2097.

**NOTE 3**

On Minerva v1.63 and Minerva v1.98, the ADATE command did not work properly - use SDATE DATE+seconds instead!

**WARNING**

ADATE will affect the time on battery backed clocks unless they are protected in some way (see PROT_DATE).

**CROSS-REFERENCE**

*DATE$* returns the current system date and time as a string, *DATE* does the same but in a less readable form - in seconds after the initial date.

*SDATE* sets the clock to an absolute date and time.

Battery backed clocks generally have their own methods of altering their date and time.

## 8.11 ADDREG

| | |
|---|---|
| Syntax | ADDREG |
| Location | TRAPS (DIY Toolkit Vol T) |

This function returns the value of the following Machine code address register following the completion of a MTRAP, QTRAP or BTRAP command.

| Command | Machine Code Register Value Returned. |
|---|---|
| MTRAP | A0 |
| QTRAP | A1 |
| BTRAP | A1 (relative to A6) - can be used by BPEEK%. |

**Example**

You could replace the ALCHP function with:

```
100 bytes=100 : REMark Number of bytes required
110 MTRAP 24,bytes,-1
120 IF DATAREG < 0 : REPORT DATAREG : REMark an error has occurred
130 IF DATAREG (1) < bytes : PRINT 'Requested area not allocated':STOP
140 base=ADDREG
```

**CROSS-REFERENCE**

*DATAREG* allows you to read machine code data registers.

See *MTRAP*, *QTRAP* and *BTRAP*.

## 8.12 ADELETE

| | |
|---|---|
| Syntax | ADELETE filename |
| Location | ATARIDOS |

This command is the same as the standard DELETE command, except that it works on Atari and IBM PS/2 format disks.

**NOTE**

You will need to pass the filename in quote marks if it includes a three letter extension preceded by a dot eg:

ADELETE “flp1_TEST.BAS”

**CROSS-REFERENCE**

See *DELETE*!

See *ADIR*, *AFORMAT*, *QACONVERT*.

## 8.13 ADIR

| | |
|---|---|
| Syntax | ADIR [#channel,] device |
| Location | ATARIDOS |

This command is the same as DIR except that it works on ATARI disks or IBM PS/2 Disks.

**CROSS-REFERENCE**

See *DIR*.

Other commands added are *ASTAT*, *ADELETE*, *ACOPY* and *AFORMAT*.

## 8.14 AFORMAT

| | |
|---|---|
| Syntax | AFORMAT device_[name] |
| Location | ATARIDOS |

This command formats the specified device in Atari disk format, giving it the specified name (if any).

As with FORMAT, this will normally format a disk to the highest possible density - however, you can force it to format a disk as single-sided by making the last character of the filename an asterisk (*).

**CROSS-REFERENCE**

See *FORMAT* and *IFORMAT*.

Other commands added are *ASTAT*, *ADELETE*, *ADIR* and *ACOPY*.

## 8.15 AJOB

| | |
|---|---|
| Syntax | AJOB jobname,priority or<br>AJOB jobnr,tag,priority or<br>AJOB job_id,priority |
| Location | Toolkit II |

This command forces the specified job (described by either its jobname, its job number and tag, or its job identification number) to be re-started at the given priority (which should be in the range 0…127 to maintain Minerva compatability - see SPJOB).

This will only work if the current priority of the given job is set to zero, in any other case, a 'Not Complete' (-1) error will be reported.

**NOTE**

It is possible that on early versions of Toolkit II, only the second syntax works.

**CROSS-REFERENCE**

*SJOB* suspends a job.

*REL_JOB* releases a job.

*SPJOB* sets the priority of a job without restarting it.

## 8.16 ALARM

| | |
|---|---|
| Syntax | ALARM hour,minutes |
| Location | Toolkit II |

This command creates a Job at low priority which makes the QL sound several beeps when the alarm time is reached and then removes itself. Naturally, this facility only works if the system clock is correct.

The hour is based on the 24-hour clock and must therefore be specified in the range 0…23 and the minutes in the range 0…59.

**Example**

How about a hourly alarm to remind you to switch off the cassette player and listen to the news on the radio?

```
100 FOR hour=8 to 18
110   ALARM hour-1,59
120 END FOR hour
```

**CROSS-REFERENCE**

Set the system clock with *SDATE*, adjust it with *ADATE*.

Alarm jobs can be killed by using *RJOB* for example.

## 8.17 ALCHP

| | |
|---|---|
| Syntax | ALCHP (space) or<br>ALCHP (space [,[jobID]]) (BTool only) |
| Location | Toolkit II, THOR XVI, BTool |

The function ALCHP allocates space bytes in the common heap and returns the start address of the memory set aside to be altered freely. This, unlike RESPR, works even if there is a task running in memory.

If ALCHP fails due to lack of available memory, then it will return 0 instead of breaking with error -3 (Out of Memory).

The BTool version of ALCHP allows an extended syntax. If space is followed by a comma ‘,’ then the allocated memory can only be removed with RECHP or CLCHP (unlike the other versions where this is done automatically with NEW and CLEAR). If the jobID is specified then not only will this be done, but the memory will also be linked to the Job identified by jobID.

**Example 1**

The following program loads two uncompressed screens from

disk into memory and shows them alternately:

```
100 adr=ALCHP(2*32768)
110 LBYTES flp1_Screen1_scr,adr
120 LBYTES flp1_Screen2_scr,adr+32768
```

(continues on next page)

(continued from previous page)

```
130 REPeat Picture_Show
140   SCRBASE adr : REFRESH : PAUSE 150
150   SCRBASE adr+32768 : REFRESH : PAUSE 150
160 END REPeat Picture_Show
```

**Example 2**

This is an alternative to the LRESPR command (although see Note 2 below):

```
100 DEFine PROCedure LALCHP (mc_file$)
110  LOCal length,adress
120  length=FLEN(\mc_file$)
130  adress=ALCHP(length)
140  LBYTES mc_file$,adress
150  CALL adress
160 END DEFine LALCHP
```

**NOTE 1**

ALCHP reserves memory in 512 byte chunks.

**NOTE 2**

Memory reserved by ALCHP is indirectly cleared by NEW, CLEAR, LOAD and LRUN (this does not apply to the Btool extended variant - see above).

**WARNING 1**

Never run device drivers in the common heap - this memory can be easily cleared, causing a spectacular crash if a device driver was stored there. This is true for other machine code, too.

**WARNING 2**

There is no checking on the parameter for ALCHP - accordingly negative values can be supplied. These are likely to lead to unexpected results and will probably crash the computer - for example, x=ALCHP(-100) crashes a JM ROM. On a Minerva ROM, values below -5 will return 0. On SMS although only values below -20 return 0, any attempt to reclaim the areas set aside with CLCHP or RECHP will crash the system.

**WARNING 3**

Since ALCHP returns 0 if there is not enough memory, you should always check the value returned by ALCHP for this before writing to the address. Otherwise, it is possible that you will be over-writing the operating system… crash!

**CROSS-REFERENCE**

The reserved parts of memory can be given back to QDOS' memory management by *RECHP* base_address or *CLCHP*.

*RESPR*, *TTALL*, *ALLOCATION* and especially *GRAB* and *RESERVE* work similar to *ALCHP*.

See *DEL_DEFB* concerning heap fragmentation.

## 8.18 ALFM

| | |
|---|---|
| Syntax | ALFM (bytes) |
| Location | SMSQ/E for Q68 |

This allocations a chunk of the available fast memory on a Q68. Returns the starting address of the allocated memory on success and an out of memory error on failure.

**Example**

```
100 x=ALFM(1024)
110 LBYTES win1_my_process,x
120 CALL x
```

**Note 1**

Very little fast memory is available with some of it being used by SMSQ/E. A freshly booted system should have about 10 kB.

**Note 2**

Like RESPR, allocated memory cannot be freed.

**CROSS-REFERENCE**

*RESPR*, *FREE_FMEM*

## 8.19 ALIAS

| | |
|---|---|
| Syntax | ALIAS old_keyword$ TO new_keyword(ALIAS_CODE) or<br>ALIAS new_keyword TO old_keyword$(SAILA_CODE) |
| Location | ALIAS (DIY Toolkit - Vol A) |

This command is similar to NEW_NAME and REPLACE.

It allows you to assign another name to machine code Procedures and Functions which are currently resident in memory. Both versions of the command are the same, except that the second variant expects you to pass the two parameters in the opposite order.

We shall deal with the first variant.

The first parameter (old_keyword$) must appear as a string and is the original name of the Procedure or Function which is to be renamed. The second parameter (new_keyword) is the new name to be used - this must not appear as a string, but simply as the actual keyword to use.

The original definition is not lost and therefore you can still use the original name to call the machine code procedure or function (as well as the new name).

If old_keyword$ does not contain the name of a machine code Procedure or Function, then either a 'Not Found' or 'Bad Name' error will be reported.

**Example**

Try the following short program:

```
10 INPUT 'Enter Your Name: '; a$
20 PRINT a$
30 ALIAS 'INPUT' TO XINPUT
40 XINPUT 'Enter My Name with XINPUT: '; s$
50 INPUT 'You can still use INPUT to Enter your Name: ';t$
60 PRINT s$ / t$
```

**NOTE 1**

Because the original definition is not lost, you can go on to assign further 'aliases' to the original name, but any attempt to assign an alias to the new name (XINPUT in the above example will give a Not Found error).

**NOTE 2**

You should not use ALIAS from within programs compiled with TURBO and SuperCharge.

**NOTE 3**

If a program compiled with TURBO or SuperCharge reports an error when you try to EXECute the program, such as 'SYS_VARS is Not Defined', you could use ALIAS from SuperBASIC to circumvent this problem, for example by using:

ALIAS 'SYSBASE','SYS_VARS'

**NOTE 4**

The new alias is not converted by this command to uppercase - that is up to you (not all keywords are in uppercase after all).

**NOTE 5**

You should not use all of the new names set with ALIAS in programs which are to be compiled with TURBO or SuperCharge if you want to make the most of those compilers. In particular, ALIASes of the following keywords will cause problems:

RESPR (unless it has been redefined to work in the common heap before you used ALIAS).

RUN, INPUT, READ, EOF, CLEAR, DIMN, STOP, NEW and various TURBO toolkit commands.

You will also lose out on optimisations on the following:

PRINT, BLOCK, CODE, CHR$, LEN, PI, PEEK, PEEK_W, PEEK_L, POKE, POKE_W and POKE_L.

**NOTE 6**

If you wish to use ALIAS for MODE and use Speedscreen, ensure Speedscreen is loaded and enabled before you use ALIAS (Speedscreen redefines MODE).

If you wish to use ALIAS for mathematical functions and use the Lightning fast maths routines, again, ensure that Lighning maths is loaded before you use ALIAS if you want the faster routines implemented by Lightning.

**NOTE 7**

If you want to use this command from within a Multiple SBASIC on SMS or a MultiBASIC on Minerva, you will need to use the variant of the command implemented in the file SAILA_CODE.

**CROSS-REFERENCE**

See also *REPLACE* and *NEW_NAME*.

*_NAME$* allows you to look at the name table.

## 8.20 ALINE

| | |
|---|---|
| Syntax | ALINE x1,y1 TO x2,y2, Colour |
| Location | Shape Toolkit |

This command quickly draws a line between the specified absolute, window independent co-ordinates, (x1,y1) and (x2,y2), on the screen. ALINE uses XOR mode, which means that the line can be removed without destroying the contents of the screen by drawing exactly the same line again. - This does however mean that the colour of the line as it appears on screen may not be the same as the specified parameter (see OVER -1).

**Example**

The procedure HAIRCROSS x,y allows you to move a cross wire around the screen with the cursor keys, to alter the values of x and y. Press <SPACE> to make x and y equal the new values, or press <ESC> to keep the old values.

```
100 DEFine PROCedure HAIRCROSS (px,py)
110  LOCal Size,Key,Stepp,old_px,old_py
120  Size=31 : old_px=px : old_py=py
140  REPeat Move_it
150    CROSS px,py
160    REPeat Wait_for_key
170      Key=KEYROW(1): Stepp=4*(KEYROW(7))+1
180      IF Key THEN EXIT Wait_for_key
190    END REPeat Wait_for_key
200    CROSS px,py
210    IF Key&&2 THEN px=px-Stepp
220    IF Key&&16 THEN px=px+Stepp
230    IF Key&&4 THEN py=py-Stepp
240    IF Key&&128 THEN py=py+Stepp
250    IF px<Size THEN px=Size
260    IF px>511-Size THEN px=511-Size
270    IF py<Size THEN py=Size
280    IF py>255-Size THEN py=Size
290    SELect ON Key
300      =64: EXIT Move_it
310      =8: px=old_px: py=old_py
320          EXIT Move_it
330    END SELect
340  END REPeat Move_it
350 END DEFine HAIRCROSS
360 :
370 DEFine PROCedure CROSS (ax,ay)
380  ALINE ax-Size,ay-Size TO ax+Size,ay+Size ,7
390  ALINE ax+Size,ay-Size TO ax-Size,ay+Size ,7
400 END DEFine CROSS
```

**NOTE 1**

ALINE assumes that the screen starts at $20000 and will therefore not work on Minerva's / Amiga QDOS's / QDOS Classic's second screen or on higher resolution displays.

**NOTE 2**

ALINE also assumes that the screen measures 512x256 pixels and cannot therefore work on higher resolution screens.

**NOTE 3**

ALINE only works in MODE 4.

**CROSS-REFERENCE**

*DRAW* has the same syntax as *ALINE* but does not work in XOR mode.*DRAW* is also able to draw lines on screens stored in memory.

*LINE* and *LINE_R* are much more flexible.

## 8.21 ALLOCATION

| | |
|---|---|
| Syntax | ALLOCATION (bytes [,taskno%,tasktag%]) |
| Location | Turbo Toolkit |

This function is very similar to RESERVE. It allocates an area in the common heap which may be associated with a specified job. If taskno% and tasktag% are not specified, then the area is linked with the current job and removed when the current job is removed.

**CROSS-REFERENCE**

*DEALLOCATE* should be used to remove the allocated area.

The taskno% and tasktag% can be found using *JOBS* or *LIST_TASKS*.

---

## 8.22 ALPHA_BLEND

| | |
|---|---|
| Syntax | ALPHA_BLEND opacity% |
| Location | SMSQ version 3.26 |

Alpha-blending is a method of drawing graphics whereby the resultant output is partly transparent – overlapping shapes and text created with BLOCK, LINE, CIRCLE, PRINT etc. will be see-through to a degree, set by a new command ALPHA_BLEND. This takes a value from 0 (fully transparent) to 255 (opaque), ALPHA_BLEND 128 will make all output half-transparent, for example.

In the past, we have only had the variations offered by the OVER command, now we can achieve some pretty exciting graphical effects for use in games, for example. Here's an example which draws three overlapping circles which are half-transparent:

```
1000 PAPER 0: CLS
1010 ALPHA_BLEND 128
1020 FILL 1: INK 2: CIRCLE 40, 50, 20
```

(continues on next page)

(continued from previous page)

```
1030 FILL 1: INK 4: CIRCLE 65, 50, 20
1040 FILL 1: INK 1: CIRCLE 50, 75, 20
1050 CSIZE 2,0: AT 10,4: INK 7: PRINT "Alpha blending!"
1060 STOP
```

In addition to the ALPHA_BLEND command, A new trap #3 with D0=$62, d1=alpha weight 0-255, d3.w=timeout and a0=channel ID allows the alpha-blend value to be set from assembler and other languages.

## 8.23 ALT

| | |
|---|---|
| Syntax | ALT |
| Location | Beuletools |

This function returns the control codes needed to switch to the alternative font (normally italics) on an EPSON compatible printer:

```
PRINT #ch,ALT
```

is therefore equivalent to:

```
PRINT #ch,CHR$(27)&"6"
```

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*, *DBL*, *ENL*, *PRO*, *SI*, *NRM*, *UNL*, *ESC*, *FF*, *LMAR*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 8.24 ALTER

| | |
|---|---|
| Syntax | ALTER 'variable' TO value |
| Location | ALTER (DIY Toolkit - Vol U) |

This command works alongside SET from the same toolkit and allows you to re-define the universal constants created with SET.

Unlike SET, the constant to be re-defined must appear in quotes as the first parameter (otherwise the value of the constant is passed to be altered by the command!!). As with SET, the constant and the value must be of the same type, otherwise an 'error in expression' will be reported.

If the constant has not previously been defined with SET, then if it is recognised for some other reason an 'In Use' error will be reported. If it is not recognised at all, then 'Not Found' will be reported.

Unlike SET, you can use ALTER from any program which is being used on the QL and therefore you can use this to update constants or possibly device names (or anything else you can invent).

**Example**

Set the following from SuperBASIC:

10 SET DEF_DRIVE$ TO 'flp1_'

If whilst using another program, the user re-defines the default device, that program can use a line such as: ALTER 'DEF_DRIVE$' TO 'win1_prog_' which will then alter the default device for all programs which read this constant.

**NOTE**

ALTER does not work on SMS.

**CROSS-REFERENCE**

See *SET*.

## 8.25 ALTKEY

| | |
|---|---|
| Syntax | ALTKEY character$,string$ [,string2$ [,string2$... ]] or<br>ALTKEY character$ or<br>ALTKEY |
| Location | Toolkit II |

This command defines a key macro which will be typed into the computer when you press the <ALT> key at the same time as the <character$> key. If more than one string follows the definition, then an <ENTER> (line feed) character is inserted between each string.

ALTKEY without any parameters deletes all previously defined ALTKEYs, whereas ALTKEY character$ will just kill the specified definition (whether there was one or not).

A line feed will not be appended to the final string unless you add a nul string to the definition.

**Example 1**

ALTKEY " ","RUN","" types in RUN <ENTER> if <ALT><SPACE> is pressed.

ALTKEY"a","flp1_" types in flp1_ when <ALT><A> is pressed.

ALTKEY removes all ALTKEY definitions.

ALTKEY "a" remove definition for <ALT><A>.

ALTKEY 1,"1000" same as ALTKEY "1","1000"

**Example 2**

There are many programs which do not support the Toolkit II default device names and sub-directories.

To avoid having to enter FLP1_Archive_Adresses_ in front of every file name, one could compile the following line, then EXECute the resultant program (using EX or EXEC) with the priority set to 1.

```
100 PRIO 1
110 REPeat Always
120   ALTKEY "p",DATAD$
130   ALTKEY "P",PROGD$
140 END REPeat Always
```

You can replace PRIO by QP QMYJOB,1 with QLiberator or PRIORITY 1 with Turbo, or SPJOB -1,1 with Toolkit II

**NOTE 1**

If character$ is an upper case letter, then you will need to press <ALT><SHIFT> and the <key> (or <ALT><key> in CAPSLOCK) to call the macro.

**NOTE 2**

The combination <ALT><ENTER> is always set aside for the last line recall (ie. when these two keys are pressed all characters typed inbetween the last two <ENTER>s are put into the keyboard buffer again).

**NOTE 3**

The Hotkey System is usually configured to type in the Hotkey Stuffer contents if <ALT><SPACE> is pressed.

<ALT> <SHIFT> <SPACE> gets previous Stuffers.

**WARNING**

If you have Hotkey System II loaded, then ALTKEY will not have any effect until you use the HOT_GO command.

**CROSS-REFERENCE**

*FORCE_TYPE* and *STAMP* allow programs to access the keyboard, *KEY* defines macros on function keys.

## 8.26 AND

| | |
|---|---|
| Syntax | condition1 AND condition2 |
| Location | QL ROM |

This combination operator combines two condition tests together and will have the value 1 if both condition1 and condition2 are true or 0 if either condition1 or condition2 are false.

A value is said to be false if it is equal to zero, anything else will cause that value to be true.

Please note the difference between this and the bitwise and operator: x&&y, which compares x and y bit by bit.

**Examples**

PRINT 1 AND 0 Returns 0

PRINT 12 AND 10 Returns 1

(compare PRINT 12&&10 which returns 8).

```
10 FOR x=1 TO 5
20   FOR y=1 TO 5
30     IF x=3 AND y>3 THEN PRINT x;' => ';y,
40   END FOR y
50 END FOR x
```

produces the following output:

3=>4 3=>5

**CROSS-REFERENCE**

*OR*, *NOT* and *XOR* are the other combination operators.

## 8.27 APOINT

| | |
|---|---|
| Syntax | APPOINT x,y,colour |
| Location | Shape Toolkit |

This command is similar to POINT, except that it uses absolute co-ordinates and plots the point in XOR mode (as with ALINE).

**NOTE**

APOINT suffers from the same problems and limitations as ALINE.

**CROSS-REFERENCE**

Use *POINT* instead!!

## 8.28 APPEND

| | |
|---|---|
| Syntax | APPEND file1,file2 |
| Location | ST/QL |

This command allows you to merge two files together by appending file2 to the end of file1.

**NOTE**

Both file1 and file2 must include the device name.

**CROSS-REFERENCE**

The THOR XVI has a special form of *COPY* which is similar to this.

## 8.29 AQCONVERT

| | |
|---|---|
| Syntax | AQCONVERT filename |
| Location | ATARIDOS |

This command takes a file which is stored on a QL Format disk and presumes that it was originally an Atari format file. It will then convert special characters in that file to QL compatible characters as well as converting any occurence of a Carriage Return character (CR) followed by a Line Feed character (LF) to a single Line Feed character LF.

**CROSS-REFERENCE**

Compare *IQCONVERT* and *QACONVERT*.

See also *ACOPY* and *QCOPY*.

## 8.30 ARC

| | |
|---|---|
| Syntax | ARC [#ch][,$x^1$,$y^1$] TO $x^2$,$y^2$,angle $^*$[[;$x^i$,$y^i$] TO $x^j$,$y^j$,angle$^j$]$^*$ |
| Location | QL ROM |

ARC causes the two points at the co-ordinates ($x^1$,$y^1$) and ($x^2$,$y^2$) to be connected with an arc. The arc is defined as the sector of the circle formed by drawing two straight lines from the two given co-ordinates to the centre of the circle, where angle is the angle (in radians) between those two lines. Therefore, angle=0 is a straight line and angle=PI, half a circle.

It therefore follows that the greater ABS(angle), the more pronounced is the curve on the arc.

Multiple arcs can be draw with the same command by adding extra sets of parameters for each additional arc. For example:

```
ARC 100,10 TO 120,40,3 TO 80,70,3
```

will actually draw two arcs, one between the points (100,10) and (120,40) with angle=3 and the second between the points (120,40) and (80,70), also with angle=3.

When drawing multiple arcs, there is actually no need for the next arc in the series to begin at the end of the previous arc, provided that a semicolon ';' is inserted between each set of parameters. For example:

```
ARC 100,10 TO 120,40,3;30,40 TO 50,60,3
```

Whether the arc is drawn clockwise or anti-clockwise depends upon two factors: If $y^1$>$y^2$ and angle>0, then the arc will be drawn anti-clockwise. Swapping the two co-ordinates or making the angle negative will force the arc to be drawn clockwise.

Co-ordinates refer to the window relative graphic co-ordinate system, which is relative to the graphic origin. The size and position of the arc also depend upon the SCALE of the window. If no first point is given then the current position of the graphic cursor is used. The graphic cursor is set to the last point of the arc on completion of the command.

**Example 1**

```
100 WINDOW 448,200,32,16: CLS: SCALE 4,-2,-2
110 FOR t=PI/16 TO 2*PI STEP PI/16
120   ARC SIN(t),COS(t) TO COS(t),SIN(t),PI*SIN(t/2)
130 END FOR t
```

**Example 2**

```
100 WINDOW 448,200,32,16: CLS: SCALE 100,0,0
110 FOR x=10 TO 90 STEP 10
120   FOR y=10 TO 90 STEP 10
130     ARC x,y TO y,x,PI/2
140   END FOR y
150 END FOR x
```

**Example 3**

```
100 POINT #2,150,50
110 FOR x=50 TO 150 STEP 20
120   ARC #2 TO x,50,PI/2
130 END FOR x
```

**NOTE 1**

On non Minerva v1.89+ ROMs, ARC does not work properly - small angles produce rubbish, wrong

co-ordinates are used and the last pixel of the arc is not always drawn. Even SMS does not cure these problems.

**NOTE 2**

An angle of 2*PI would form a complete circle and cannot be drawn, therefore the maximum value for ABS(angle) is a value just less than 2*PI.

**NOTE 3**

On some ROM versions, the command does not check that the TO separator is present - however, SMSQ/E (at least) does and therefore some programs may fail if used under SMSQ/E and they have used a comma instead of TO.

**WARNING**

Some QDOS implementations of this command can corrupt the hard disk drive in some obscure circumstances. Get Minerva or SMSQ/E to be safe!!

**CROSS-REFERENCE**

*ARC_R* works in exactly the same way as *ARC* but uses a relative co-ordinate system, where the origin is the current position of the graphic cursor.

*SCALE* sets the graphic origin and also the size of the window.

## 8.31 ARC_R

| | |
|---|---|
| Syntax | ARC_R [#ch][,$x^1$,$y^1$] TO $x^2$,$y^2$,angle *[[;$x^i$,$y^i$] TO $x^j$,$y^j$,angle$^j$]* |
| Location | QL ROM |

This command draws an arc relative to the current graphic cursor. This means that rather than the co-ordinates (x,y) being relative to the graphic origin, they are relative to the current graphic cursor. Arcs are however still affected by the current SCALE.

Each set of co-ordinates used in the ARC_R command moves the graphic cursor, which means for example that ($x^1$,$y^1$) is relative to the graphic cursor when ARC_R is first called, whereas ($x^2$,$y^2$) is relative to ($x^1$,$y^1$).

**Example 1**

A short program to draw several equi-distant arcs using ARC_R:

```
100 WINDOW 448,200,32,16:SCALE 100,0,0
110 PAPER 0:INK 4:CLS
120 ARC 20,20 TO 90,20,PI/4
130 FOR i=1 TO 4
140   ARC_R 0,10 TO -70,0,-PI/4
150   ARC_R 0,10 TO 70,0,PI/4
160 END FOR i
```

**Example 2**

The same routine, but altered to use ARC:

```
100 WINDOW 448,200,32,16:SCALE 100,0,0
110 PAPER 0:INK 4:CLS
120 ARC 20,20 TO 90,20,PI/4
130 FOR i=30 TO 100 STEP 10
140   ARC 20,i TO 90,i,PI/4
150 END FOR i
```

**CROSS-REFERENCE**

The graphic cursor is moved with commands such as *POINT*, *ARC*, *CIRCLE* and *LINE*.

Please also see *ARC*.

## 8.32 ARCOSH

| | |
|---|---|
| Syntax | ARCOSH (x) |
| Location | Hyper |

This function returns the arc hyperbolic cosine of the specified value, that is to say it will return the value which must be passed to the hyperbolic cosine to return the given result, so:

```
COSH ( ARCOSH ( x )) = x
```

The ARCOSH function can be expressed as a combination of SuperBASIC keywords: it's the same as:

```
LN(x + SQRT(x*x-1)).
```

**CROSS-REFERENCE**

See *ACOS*, *ASIN*, *ACOT*, *ATAN*, *COSH*, *ARCOTH*, *ARSINH* and *ARTANH*.

## 8.33 ARCOTH

| | |
|---|---|
| Syntax | ARCOTH (x) |
| Location | Hyper |

This function returns the arc hyperbolic cotangent of the specified value ie.

ARCOTH ( COTH ( x )) = x

Or to keep it simple, it can be returned with the equivalent expression LN((x+1) / (x-1)) / 2

**CROSS-REFERENCE**

See *ACOT*, *ARCOSH*, and *ARTANH*.

## 8.34 ARSINH

| | |
|---|---|
| Syntax | ARSINH (x) |
| Location | Hyper |

This function is the arc hyperbolic sine (ie. the complementary function to SINH).

The SuperBASIC expression:

LN(x + SQRT(x*x-1))

gives the same value.

**CROSS-REFERENCE**

See *ASIN*, *ARCOSH*, and *ARCOTH*.

## 8.35 ARTANH

| | |
|---|---|
| Syntax | ARTANH (x) |
| Location | Hyper |

The function ARTANH returns the value which must be passed to TANH to give the specified result, so:

TANH ( ARTANH ( x )) = ARTANH ( TANH ( x )) = x

ARTANH(x) could be replaced by: LN((1+x) / (1-x)) / 2

**CROSS-REFERENCE**

See *ATAN*, *ARCOTH*, and *ARSINH*.

## 8.36 ASIN

| | |
|---|---|
| Syntax | ASIN (x) |
| Location | QL ROM |

This function calculates the arc-sine (in radians) which is the opposite of the sine function, ie:

x = SIN ( ASIN ( x )) = ASIN ( SIN ( x ))

The only valid values of x are in the range -1…1. This means that the range of angles supported by this command are -PI/2…PI/2. A negative angle means that the hypotenuse appears below the base line of the triangle (you must therefore always bear the orientation of the screen in mind when using this command).

**Example**

Given that there are two points on the screen at (10,20) and (100,75), find the angle of the line between those two points (from the horizontal):

```
100 PRINT CALC_ANGLE(10,20 TO 100,75)
110 STOP
120 :
200 DEFine FuNction CALC_ANGLE(x1,y1,x2,y2)
210  LOCal Distan, Radian_angle
220  Distan = SQRT((x2-x1)^2 + (y2-y1)^2)
230  Radian_angle = ASIN((y2-y1) / Distan)
```

(continues on next page)

(continued from previous page)

```
240  RETurn DEG(Radian_angle)
250 END DEFine
```

**MINERVA NOTE**

On a Minerva you can replace line 220 with: 220 Distan = ABS(x2-x1,y2-y1)

**CROSS-REFERENCE**

*ACOS*, *ATAN*, *ACOT* are other arc functions, *SIN*, *COS*, *TAN* and *COT* their relatives.

Please also see the Mathematics section of the Appendix.

Compare *ARSINH*.

## 8.37 ASK

| | |
|---|---|
| Syntax | ASK ([#wind,] question$) |
| Location | BTool |

ASK is a function which prints the question$ (plus a question mark (?) if this was not found at the end of the string), enables the text cursor and reads the keyboard. If the next key pressed is <Y> (for Yes), <J> (for Ja) or <N> (for No or Nein) then ASK will disable the cursor, echo the key next to the prompt and return 1 if either <Y> or <J> was pressed, or 0 if <N> was pressed. If any other key is pressed, ASK will BEEP and try again.

**Example**

In early computer days, this was a classical game which needed a hundred lines on a (modern at the time) programmable pocket calculator:

```
100 CLS: x1 = 0: x2 = 100
110 PRINT "I am going to find out a number"
120 PRINT "from"!x1!"to"!x2!"which only you know."\\
130 REPeat find_out
140   PRINT x1;"..";x2
150   x = (x2+x1) DIV 2
160   ok = ASK("Is it "&x)
170   IF ok THEN EXIT find_out
180   IF x1 = x2 THEN PRINT "You are cheating.": STOP
190   large = ASK(x&" too large")
200   IF large THEN x2 = x-1: ELSE x1 = x+1
210 END REPeat find_out
220 PRINT "Yippee, I found it."
```

**NOTE**

ASK is set up for 'yes' and 'no' in English and 'ja' and 'nein' in German. For other languages where 'yes' is not usually connected with <Y>, eg. 'oui' in French or 'si' in Spanish, you will need to write your own routine.

**CROSS-REFERENCE**

*CUR*, *REPLY*.

## 8.38 ASTAT

| | |
|---|---|
| Syntax | ASTAT [#channel,] |
| Location | ATARIDOS |

This command is similar to ADIR except that it also provides extra information, such as the length of each file, the update time and any marks folder.

**CROSS-REFERENCE**

See *ADIR*. *WSTAT* is similar on QL Format disks.

Other commands added are *ADELETE*, *ACOPY* and *AFORMAT*.

## 8.39 AT

| | |
|---|---|
| Syntax | AT [#ch,] row, column or<br>AT [#ch,] column,row (pre AH ROMs only) |
| Location | QL ROM |

This command sets the current print position in the given window (default #1) to the given row and column number. The top left hand corner of any window is always the position (0,0), however, the maximum values of the row and column numbers depends on both the size of the window and the current character size. Anything outside of this will give the error 'Out of Range' (-4).

Unlike the PRINT parameter TO, this command does not print any spaces on screen, thus allowing you to place text precisely on screen without deleting any other parts of the screen.

Unfortunately for users who learnt to program on early versions of Sinclair BASIC (on the ZX81 or Spectrum), this command is implemented differently.

Some implementations of BASIC allow you to set the print position from within the PRINT command, for example:

PRINT AT 3,5;'Hello'

On the QL, you would need the line:

AT 3,5: PRINT 'Hello'

**Example**

A program which uses the AT command to create an interesting effect on screen. This will not work on pre JS ROMs as it relies upon the WHEN ERRor command:

```
1000 WHEN ERRor
1010   IF ERR_OR THEN dir1=-dir1: y=y-2: RETRY 1070
1020 END WHEN
1025 :
1030 MODE 4:WINDOW 448,200,32,16:CSIZE 0,0
1040 x=0: dir1=1
1050 FOR y=0 TO 63
1060   AT x,y:PRINT 'Sinclair QL'
```

(continues on next page)

(continued from previous page)

```
1070   x=x+dir1
1080 END FOR y
```

**NOTE**

On early QL ROMs (pre AH), the parameters were mixed up meaning that the syntax was AT column,row. This can of course create many problems in uncompiled SuperBASIC, however, there should not be many of these machines left.

If you do have one of these early machines, it is recommended that you do update the ROM.

**CROSS-REFERENCE**

*CSIZE* sets the current character size for the given window

*WINDOW* alters the physical size of a given window.

*CURSOR* allows you to set the print position more exactly.

*PRINT* actually prints things on screen at the current print position.

*VER$* allows you to check the ROM version.

Also see *LEFT*.

## 8.40 ATAN

| | |
|---|---|
| Syntax | ATAN (x) or<br>ATAN (x,y) (Minerva and SMS only) |
| Location | QL ROM |

The function ATAN, is the arc-tangent function, that is to say the inverse of the tangent function (TAN in SuperBASIC).

TAN ( ATAN ( x )) = x

for all values of x, but due to the fact that TAN works on periods; ATAN ( TAN ( x )) = x

is only true for where: -PI/2 < x < PI/2.

A negative angle indicates that the hypotenuse appears below the base line of the triangle, and it is therefore important to bear in mind the orientation of the screen when using this command.

**NOTE 1**

Because trigonometrical functions are calculated using polynomial approximations, large parameters can produce small errors.

For example, on all implementations:

PRINT TAN (ATAN ( 123456 ))

gives 123461.2 instead of 123456.

The maximum error rises in direct proportion to the parameter for the above example.

**NOTE 2**

There is a very obscure bug contained in the code for ATAN which means that the command may crash on non-Minerva ROMs if used in a program which is longer than 32K.

**MINERVA NOTE**

ATAN can accept two parameters. If you specify two parameters then ATAN(x,y) will give the angle from the origin to the point (x,y). This is actually the same as ATAN(y/x), although it does also cater for when x=0 which would otherwise give an overflow error.

This variant also supports a full circle, for example the following can be used to calculate the bearing travelled (with 0 degrees being north), given that you have moved x miles east (or west if x<0) and y miles north (or south if y<0):

```
100 DEFine PROCedure BEARING (x,y)
110 direction=DEG (ATAN (y,x))
120 IF x>=0: RETurn direction: ELSE RETurn 360+direction
130 END DEFine
```

The need for line 120 is because the value returned by ATAN is in the range -PI … PI (which converts to -180 … +180 degrees) - the value returned needs to be in the range 0 … 360. Note that x and y are swapped around in line 110 - this is to circumvent the problem that a bearing of 0 is north, whereas in the mathematical functions, a zero is taken to be horizontal.

**SMS NOTE**

The ATAN function has been extended to be the same as on Minerva, although the range of values it returns have been made into four quadrant results (as with ATN2 ), so that for ATAN(x,y) if x>0, the result is now in the range -PI/2…PI/2 instead of the usual 0…PI.

**CROSS-REFERENCE**

*TAN*, *ATN*, *ATN2* and *ARTANH*. Also please refer to the Mathematics section in the Appendix.

## 8.41 ATARI

| | |
|---|---|
| Syntax | ATARI |
| Location | Beuletools |

On the Atari QL-Emulator, this command switches to Atari mode. Naturally, on other machines, it has no effect. It will also fail if a QL ROM was found at the start address of the ROM-TOS ($FC0000) - it is possible to load QDOS to that address.

**NOTE**

The FN Toolkit (pre v1.04) contained a function of the same name which had a different effect - this has now been renamed QuATARI (see below).

**WARNING**

This command will most probably fail on the latest ST/QL drivers.

**CROSS-REFERENCE**

See *QuATARI*.

## 8.42 ATARI_EXT

| | |
|---|---|
| Syntax | ATARI_EXT |
| Location | ATARI_REXT (v2.15+) |

The Atari QL-Emulators come with the additional toolkits, ATARI_REXT and ATARIDOS.

This command is used to enable various commands in the ATARI_REXT toolkit as well as the sound extensions (such as BELL).

It therefore replaced the original SND_EXT command.

**WARNING**

ATARI_REXT pre v2.37 may crash SMS.

**CROSS-REFERENCE**

See *TK2_EXT* and *Beule_EXT*.

See also *SND_EXT*.

## 8.43 ATN

| | |
|---|---|
| Syntax | ATN (x) |
| Location | Math Package |

This function is the same as the original QL ROM variant of ATAN.

**NOTE**

ATN has been implemented to make porting programs written in other BASIC dialects easier.

**CROSS-REFERENCE**

See *ATAN*.

## 8.44 ATN2

| | |
|---|---|
| Syntax | ATN2 (x,y) |
| Location | Math Package |

ATN2 calculates ATAN(x/y) but expands the result from 0…PI to -PI…PI which allows you to convert cartesian and polar co-ordinates in both directions without loss of information.

**Example**

Run this graphics demonstration and you will understand the advantage of ATN2 and the difference from ATAN:

```
100 WTV 4: SCALE 4,-3,-2: INK 7
110 PAPER 0: OVER -1: CLS
120 radius=1.5: reso=128
130 FOR angle0=PI/reso TO 2*PI STEP PI/reso
140   x0=radius*COS(angle0): y0=radius*SIN(angle0)
150   angle1 = ATAN(y0/x0)
160   x1=radius*COS(angle1): y1=radius*SIN(angle1)
170   angle2 = ATN2(x0,y0)
180   x2=radius*COS(angle2): y2=radius*SIN(angle2)
190   ARRAYS: PAUSE 2: ARRAYS
200 END FOR angle0
210 :
220 DEFine PROCedure ARRAYS
230  INK 3
240  FILL 1: CIRCLE 1.25*x0,1.25*y0,5E-2: FILL 0
250  IF x1==x2 AND y1==y2 THEN
260    INK 7: LINE x1/5,y1/5 TO x1,y1: INK 5
270    CURSOR x1,y1,0,0: PRINT "ATAN/ATN2"
280  ELSE
290    INK 7: LINE x1/5,y1/5 TO x1,y1: INK 5
300    CURSOR x1,y1,0,0: PRINT "ATAN"
310    INK 7: LINE x2/5,y2/5 TO x2,y2: INK 5
320    CURSOR x2,y2,0,0: PRINT "ATN2"
330  END IF
340  angle=INT(DEG(angle0))
350  CURSOR 0,0,-3*LEN(angle),-5: PRINT angle
360 END DEFine ARRAYS
```

**CROSS-REFERENCE**

*ATAN* which is the same on Minerva and SMS.

## 8.45 AUTO

| | |
|---|---|
| Syntax | AUTO [start_number][,step] |
| Location | QL ROM |

This command automatically creates line numbers in the command line (#0) to assist in entering SuperBASIC programs. It would normally only be entered as a direct command (although you can include it in a program line, the line numbers will not be generated until the program has finished its work).

Once entered, you will be presented with the first line start_number (default 100) - if this line already exists in the program, then the existing line will be presented. Otherwise, you will only see the current line number. Pressing the up and down arrow keys will move you to the previous line or the next line (respectively) in the program, although if there is no previous (or next) line, then you will exit the AUTO mode. However, if you press the Enter key, the next line number will be generated by adding step (default 10) to the current line number.

If you wish to escape this sequence, press the Break key <CTRL><SPACE>.

**Example 1**

Generating program lines: AUTO 1000,10

generates lines 1000,1010,1020,1030,… AUTO 10

generates lines 10,20,30,40,…. AUTO ,5

generates lines 100,105,110,115,…

**Example 2**

Adding line numbers to a numberless boot program: AUTO 100,10: MERGE flp1_boot

**NOTE 1**

A step value of zero returns 'Bad Parameter' (-15). You can however achieve this by using EDIT start_number instead.

**NOTE 2**

Did you realise that AUTO 200,10 is the same as EDIT 200,10 ?

**NOTE 3**

On non-Minerva ROMs AUTO uses the same routine as RENUM to check its parameters, which means that you can specify a start_line and an end_line, although they do nothing. For example:

AUTO 100 TO 1000;1000,20

would create lines 1000,1020,1040,…

**NOTE 4**

The maximum line number is 32767 - trying to use a higher line number will cause an overflow error.

**NOTE 5**

If start_number and step are not integer numbers, they will be rounded either up or down to the nearest integer (compare INT).

**SMS NOTE**

On current versions of SMS AUTO has been re-coded to be the same as ED, therefore it will not allow a second parameter, and merely places you in ED mode with the cursor at the specified start line number.

**CROSS-REFERENCE**

Please refer to *EDIT* which is very similar.

*DLINE* allows you to delete SuperBASIC lines.

## 8.46 AUTO_DIS

| | |
|---|---|
| Syntax | AUTO_DIS |
| Location | Super Gold Card, Gold Card v2.67+ |

The Super Gold Card allows you to automatically start-up the QL (overcoming the need to press F1 or F2 on the title screen), and also automatically start up Toolkit II.

This command switches off these features.

**NOTE 1**

On Minerva these commands only dictate whether Toolkit II should automatically be started up, as Minerva contains its own auto-boot code.

**NOTE 2**

These commands have no effect under SMSQ/E which already includes Toolkit II and does not show a start-up screen.

**CROSS-REFERENCE**

See*AUTO_TK2F1* and *AUTO_TK2F2* also.

## 8.47 AUTO_TK2F1

| | |
|---|---|
| Syntax | AUTO_TK2F1 |
| Location | Super Gold Card, Gold Card v2.67+ |

The Super Gold Card allows you to automatically boot up the machine whenever it is switched on or reset.

This command enables this auto-booting (starting the machine up in Monitor mode) and also ensures that Toolkit II is initialised as soon as the machine is switched on. The status set by this command is remembered by the Super Gold Card even when the power is disconnected.

**CROSS-REFERENCE**

See also *AUTO_DIS* and *AUTO_TK2F2*.

*TK2_EXT* is needed to initialise Toolkit II if this command has not been used.

## 8.48 AUTO_TK2F2

| | |
|---|---|
| Syntax | AUTO_TK2F2 |
| Location | Super Gold Card, Gold Card v2.67+ |

This command is the same as *AUTO_TK2F1* except that the machine is started up in F2 TV mode.

**CROSS-REFERENCE**

See *AUTO_TK2F1*.

## 8.49 A_BLANK

| | |
|---|---|
| Syntax | A_BLANK [minutes] |
| Location | ST/QL (Pre v2.24) |

This command creates a small job which blanks out the screen if a key has not been pressed for a specified number of minutes (default 5).

This command is useful, because if a very bright picture is drawn on screen (eg. a white line on black paper), and the screen does not alter, this can lead to what is known as 'burn in' when the monitor screen becomes permanently marked with the 'ghost' of the picture. This does not tend to happen very often

nowadays, but in the past, monitors tended to become unuseable as more and more of their screen became covered with these 'ghosts'.

**NOTE**

This command will only work within the Pointer Environment.

**CROSS-REFERENCE**

*BLS* is a similar function under SERMouse.

## 8.50 A_EMULATOR

| | |
|---|---|
| Syntax | A_EMULATOR |
| Location | ATARI_REXT v2.22+ |

This function returns a number to signify the type of ST/QL EMulator which is being used with the Atari computer. The value returned may be one of the following:

- 0 - QL Emulator (the original QL Emulator)
- 1 - Extended-Mode4 Emulator
- 2 - QVME Emulator

**NOTE 1**

This will only work with Level E-20 of the Drivers or later.

**NOTE 2**

It is impossible to tell whether the original QL Emulator supports MODE 8 or not.

**NOTE 3**

You can also use DISP_TYPE to find out the Emulator type.

**CROSS-REFERENCE**

See also *PROCESSOR* and *MACHINE*.

## 8.51 A_MACHINE

| | |
|---|---|
| Syntax | A_MACHINE |
| Location | ATARI_REXT v2.22+ |

This function is the same as MACHINE.

**CROSS-REFERENCE**

See*MACHINE* and also *A_EMULATOR*.

## 8.52 A_OLDSCR

| | |
|---|---|
| Syntax | A_OLDSCR |
| Location | ATARI_REXT (v2.27+) |

A lot of software (mainly non-pointer driver programs), and some of the toolkits covered by this book, written for the Sinclair QL in the past always assumed that the QL screen would appear at the memory location 131072 ($20000 in hexadecimal).

These programs and toolkits will not work properly (if at all) on the QVME board or some higher resolution screens. One of the solutions to this is to use the command A_OLDSCR which forces ST/QL Emulators to set up a Job copying the QL's screen as stored at 131072 (onwards) to the real display screen 20 times a second. This obviously slows down the operation of the computer and thus if possible, a new version of the software should be produced / obtained.

As from v2.30, this command will not affect the display speed as much on a machine fitted with a blitter chip.

**NOTE 1**

This command cannot fix the problem with programs and toolkits which assume that the QL's display is 512x256 pixels.

**NOTE 2**

This command reports 'Not Implemented' on other ST-QL Emulators.

**NOTE 3**

SuperBasic (Job 0) must be the only Job running on the machine when this command is issued, otherwise the error 'Not Complete' is reported.

**NOTE 4**

If you try to use this command after it has already been issued, the errror 'Already Exists' is reported.

**CROSS-REFERENCE**

*SCREEN* can be used to find the screen address.

*SCR_SIZE* can be used to set the resolution of the display - much software will insist that this is set to 512x256 pixels also.

## 8.53 A_PROCESSOR

| | |
|---|---|
| Syntax | A_PROCESSOR |
| Location | ATARI_REXT v2.22+ |

This function is the same as PROCESSOR.

**CROSS-REFERENCE**

See *PROCESSOR*!

## 8.54 A_RDATE

| | |
|---|---|
| Syntax | A_RDATE |
| Location | ATARI_REXT (v2.10+) |

This command sets the QL's internal clock to the date and time contained in the battery-backed clock on the ST (if available).

**NOTE**

Before v2.28, this command did not support the TT's battery backed clock.

**CROSS-REFERENCE**

See *A_SDATE*.

## 8.55 A_SDATE

| | |
|---|---|
| Syntax | A_SDATE year, month, day, hour, minute, second |
| Location | ATARI_REXT |

The Atari ST has a built in battery-backed clock which maintains the time whilst the machine is switched off. This time is automatically copied across to the Emulator's own internal clock when the Atari ST is started up. However, it can be necessary to alter the Atari's battery backed clock.

This is achieved by using the command A_SDATE in exactly the same way as you would use SDATE to set the internal clock.

**NOTE 1**

Before v2.19 of Atari_rext (and in v2.23), this command will not alter the Emulator's internal clock until the Atari is reset.

**NOTE 2**

Before v2.29, this command did not support the TT's battery backed clock.

**CROSS-REFERENCE**

See *SDATE*.

*A_RDATE* will set the internal clock to the same date and time as the battery backed clock.

## 8.56 A_SPEED

| | |
|---|---|
| Syntax | A_SPEED value |
| Location | ATARI_REXT |

Due to the enhanced hardware on which the ST/QL Emulator is running, you may find that as with the QXL, Super Gold Card and Gold Card, some programs run too quickly. The command A_SPEED allows you to slow the Emulator down so that you can use these programs. value must be in the range 0..7.

0 allows the Emulator to run at full speed, whereas 7 makes it run very slowly.

### CROSS-REFERENCE

*SLUG* is very similar.

CHAPTER

NINE

# KEYWORDS B

## 9.1 BASIC

| | |
|---|---|
| Syntax | BASIC [(offset)] |
| Location | BTool |

The function BASIC is identical to BASICP except that if no parameter is supplied, the function BASIC will return the base address of the SuperBASIC program area.

**NOTE**

Although this function is written in such a way that it can be used from within compiled programs to access SuperBASIC variables, it cannot access MultiBASIC variables on Minerva nor SBASIC variables on SMS and will always return a value representing the location of the equivalent SuperBASIC variable.

**WARNING**

You should use commands such as the extended PEEK or POKE provided by Minerva and SMS to read or set the tables pointed to by the values returned by this function, as the SuperBASIC tables can move when tasks are started up or removed from memory.

**CROSS-REFERENCE**

See *BASICP* and *BASIC_W*.

## 9.2 BASICP

| | |
|---|---|
| Syntax | BASICP (offset) |
| Location | TinyToolkit |

This function returns an internal pointer (address) used by the SuperBASIC interpreter. offset must be non-negative and a multiple of 2 (up to a maximum of HEX('64').

Refer to system documentation for more information.

**Example**

PRINT BASICP(16)

returns the start address of the current SuperBASIC program in memory.

PRINT BASICP(32)

returns the start address of the SuperBASIC name list.

**NOTE**

This suffers from the same problem as BASIC.

**CROSS-REFERENCE**

*BASIC_B*, *BASIC_L*, *BASIC_W*. See *NEW_NAME* for a useful example!

*BASIC_POINTER* is the same as this function.

*_NAME$* and *BASIC_NAME$* allow you to access the SuperBASIC name list safely.

## 9.3 BASIC_B

See *BASIC_L* below.

## 9.4 BASIC_W

See *BASIC_L* below.

## 9.5 BASIC_L

| | |
|---|---|
| Syntax | BASIC_B (offset) and<br>BASIC_W (offset)<br>BASIC_L (offset) |
| Location | TinyToolkit, BTool, Turbo Toolkit (BASIC_L only) |

These three functions are modified version of PEEK which return values at memory locations in the SuperBASIC system variables, which are used for storage by the SuperBASIC interpreter.

BASIC_B returns bytes, BASIC_W words and BASIC_L long words.

**Example**

Although additional information about internal machine structures is necessary to make full use of these functions, some simple tasks can be performed without this knowledge, for example:

PRINT BASIC_W ( 1076 )

gives the first line number of a program in memory - this enables a machine code program to check if a program is actually loaded in the machine. The value returned by this example will always be zero from the interpreter.

```
100 IF NOT BASIC_W (1076) THEN
110   PRINT "No SuperBASIC program loaded"
120 END IF
```

**NOTE**

These functions generally suffer the same problem as BASIC.

A file called TurboFix_bin can be used to allow BASIC_L to access Minerva MultiBASIC and SMS SBASIC variables. Some early versions of TurboFix_bin have bugs in it. Beware that not all versions of this file supports SMS SBASICs.

**CROSS-REFERENCE**

*PEEK*, *PEEK_W*, *PEEK_L*, *BASICP*.

*BASIC_B%*, *BASIC_W%*, *BPEEK%*, *BPEEK_W%* and *BPEEK_L* are similar.

See also *BASIC_F* and *PEEK_F*.

The SuperBASIC variables appear in the QDOS/SMS Reference Manual (Section 18.3)

## 9.6 BASIC_B%

See *BASIC_F* below.

## 9.7 BASIC_W%

See *BASIC_F* below.

## 9.8 BASIC_F

| | |
|---|---|
| Syntax | BASIC_B% (offset) and<br>BASIC_W% (offset) and<br>BASIC_F (offset) |
| Location | Turbo Toolkit, BTool, Turbo Toolkit (BASIC_L only) |

The functions BASIC_B% and BASIC_W% are similar to BASIC_B and BASIC_W. BASIC_F is a further function which can be used to return a floating point number stored as six bytes starting at the specified offset within the SuperBASIC system variables.

**NOTE**

A file called TurboFix_bin can be used to allow these functions to access Minerva MultiBASIC / SBASIC variables. Some early versions of TurboFix_bin have bugs in it. Beware that not all versions of this file supports SMS SBASICs.

**CROSS-REFERENCE**

Refer to *BASIC_B* and *BASIC_W*. *PEEK_F* is similar to *BASIC_F*.

## 9.9 BASIC_INDEX%

| | |
|---|---|
| Syntax | BASIC_INDEX% (name$) |
| Location | Turbo Toolkit |

This function is similar to LOOKUP%, except it does not suffer with any problems under SMS.

If the specified name$ does not exist, -12 is returned. -7 is returned if there is some mismatch between table entries.

**NOTE**

A file called TurboFix_bin can be used to allow BASIC_INDEX% to access the Minerva MultiBASIC and SMS SBASIC name tables. Some early versions of TurboFix_bin have bugs in it. Beware that not all versions of this file supports SMS SBASICs.

**CROSS-REFERENCE**

Refer to *LOOKUP%* and *BASIC_NAME$*.

## 9.10 BASIC_NAME$

| | |
|---|---|
| Syntax | BASIC_NAME$ (index) |
| Location | Turbo Toolkit |

This function is exactly the same as _NAME$.

If the specified index is greater than the maximum name table entry, a bad parameter error is returned. If it is smaller than 0, an error may be generated, or junk may be returned.

**NOTE**

A file called TurboFix_bin can be used to allow BASIC_NAME$ to access the Minerva MultiBASIC and SMS SBASIC name tables. Some early versions of TurboFix_bin have bugs in it. Beware that not all versions of this file supports SMS SBASICs.

**CROSS-REFERENCE**

Refer to *_NAME$* and *BASIC_INDEX%*.

## 9.11 BASIC_POINTER

| | |
|---|---|
| Syntax | BASIC_POINTER (offset) |
| Location | Turbo Toolkit |

This function is exactly the same as BASICP.

**NOTE**

A file called TurboFix_bin can be used to allow BASIC_POINTER to access the Minerva MultiBASIC and SMS SBASIC name tables. Some early versions of TurboFix_bin have bugs in it. Beware that not all versions of this file supports SMS SBASICs.

**CROSS-REFERENCE**

Refer to *BASICP*.

## 9.12 BASIC_TYPE%

| | |
|---|---|
| Syntax | BASIC_TYPE% (offset) |
| Location | Turbo Toolkit |

This function looks at the entry in the SuperBASIC name table for Job 0 at the specified offset and returns its type in accordance with the following values:

- 0 .... no type
- 1 .... string
- 2 .... floating point
- 4 .... integer

If the specified offset is greater than the maximum name table entry, a bad parameter error is returned. If it is smaller than 0, an error may be generated, or junk may be returned.

**NOTE**

A file called TurboFix_bin can be used to allow BASIC_TYPE% to access the Minerva MultiBASIC and SMS SBASIC name tables. Some early versions of TurboFix_bin have bugs in it. Beware that not all versions of this file supports SMS SBASICs.

**CROSS-REFERENCE**

*TYPE* is similar.

See also *BASIC_NAME$*.

## 9.13 BAT

| | |
|---|---|
| Syntax | BAT |
| Location | Beuletools |

This command forces the command string defined with BAT_USE to be typed into the command line (#0). No parameters are allowed. BAT will work okay in Minerva's MultiBASICs, SMS's SBASICs and even if #0 has been redefined.

**CROSS-REFERENCE**

See *BAT_USE* for an example.

Refer to *TYPE_IN* also.

## 9.14 BAT$

| | |
|---|---|
| Syntax | BAT$ |
| Location | Beuletools |

This function returns the current string (if any) which has been set up with the BAT_USE command.

**CROSS-REFERENCE**

See *BAT_USE* and *BAT* for more details.

## 9.15 BAT_USE

| | |
|---|---|
| Syntax | BAT_USE batch$ |
| Location | Beuletools |

This command is used to specify a command string containing SuperBASIC keywords which will be typed into the command line (#0) when the command BAT is issued. The string may be up to 128 characters long. You may add CHR$(10) to the end of the string in order to emulate an <ENTER> keypress (as in the example below).

**Example**

BAT_USE "PAPER 3: INK 7: PAPER#2,3: PAPER#2,3: INK#2,7: WMON 4: BORDER 1,0: BORDER#2,0" & CHR$(10)

The command BAT will now reset the standard start-up windows.

**CROSS-REFERENCE**

*BAT* executes the batch string set with *BAT_USE*.

See *FORCE_TYPE*,*STAMP* and *TYPE_IN* also.

*DO* allows batch files of any size to be executed.

## 9.16 BAUD

| | |
|---|---|
| Syntax | BAUD bps or<br>BAUD [port,] bps(SMS and ST/QL only) |
| Location | QL ROM |

The serial port(s) use a certain speed to communicate with printers, modems, other computers, interfaces etc. This speed is set with BAUD. The only values allowed are set out below, any other value for bps will produce an error. The unit of the parameter is bits per second.

BAUD will set the same output and input baud rate for both serial ports.

| Bits/Sec | Bytes/Sec | Time/32Kb |
|---|---|---|
| 75 | 9.375 | 58 min, 15 sec |
| 300 | 37.5 | 14 min, 34 sec |
| 600 | 75 | 7 min, 17 sec |
| 1200 | 150 | 3 min, 38 sec |
| 2400 | 300 | 1 min, 49 sec |
| 4800 | 600 | 55 sec |
| 9600 | 1200 | 27 sec |
| 19200 | 2400 | 14 sec |

**NOTE 1**

The effect of BAUD 19200 depends on the hardware. On standard QLs the serial port can only send data at that baud rate and tends to be affected by the QL's sound chip.

**NOTE 2**

On a standard QL without Minerva the actual baudrate is slightly lower than that stated above.

**NOTE 3**

In practice, data is compressed and transmitted with transfer protocols (to reduce transmission errors), so the above transmission times refer to the actual speed of the hardware, not the amount of data.

**NOTE 4**

The standard QL cannot safely handle the input of data at baud rates greater than 1200.

**NOTE 5**

A modified co-processor Hermes which replaces the 8049 chip by a 8749 is available, which allows independent input baud rates and (if Minerva v1.95+ is present) independent output baud rates as well as fixing all mentioned problems for QLs and AURORA boards. The more expensive version of Hermes (SuperHermes) also provides three additional low speed RS232 input ports (supporting 30 to 1200 bps) and a high speed RS232 two way serial port (supporting up to 57,600 bps, which equates to 4800 characters per second).

**NOTE 6**

On a QXL board without SMS v2.57+, a BAUD command would not have immediate effect if a serial channel was open - it waited until you closed the channel.

**NOTE 7**

It is possible to connect a mouse to a QL through the standard serial port. Although the mouse operates at 1200 baud, you can use the mouse alongside a printer (or modem) either with the assistance of Hermes or by configuring the mouse software to de-activate whilst the higher baud rate is in use.

**THOR XVI NOTES**

The THOR XVI allows the following additional baud rates:

| Bits/Sec | Bytes/Sec | Time/32Kb |
|---|---|---|
| 110 | 13.75 | 39 min, 43 sec |
| 134.5 | 16.8125 | 32 min, 29 sec |
| 150 | 18.75 | 29 min, 08 sec |
| 1800 | 225 | 2 min, 26 sec |

Independent baud rates may also be used on output and input channels when the channel is opened by using an extended device name.

**MINERVA NOTES**

Minerva v1.93+ now enables you to set different output baudrates for ser1 and ser2 - if you want different input baudrates for the two ports, you will need Hermes (see above). Unfortunately, this enhancement will only work on QLs without Hermes if both ports are output only.

If you want to disable the ability to handle different output baud rates, do so with the command: POKE !124 !49,2

In order to set the two baudrates independently, BAUD will now accept additional values in the range -1 to -128. This is calculated by looking at the following table, working out which features you will need and adding the values accordingly to -128:

| Value to Add | Effect |
|---|---|
| 64 | Alters ser2 baudrate (ser1 is default) |
| 16 | Prevents standard BAUD command from altering baudrate on this port |
| 7 | Selects BAUD 75 on this port |
| 6 | Selects BAUD 300 on this port |
| 5 | Selects BAUD 600 on this port |
| 4 | Selects BAUD 1200 on this port |
| 3 | Selects BAUD 2400 on this port |
| 2 | Selects BAUD 4800 on this port |
| 1 | Selects BAUD 9600 on this port |
| 0 | Selects BAUD 19200 on this port |

Please only try to add one baud rate value!!

**Minerva Examples**

BAUD -128

sets the baud rate for ser1 output at 19200. ser2 is unaffected.

BAUD -47

fixes the baud rate for ser2 output at 9600. ser1 is unaffected (-47 = -128+64+16+1).

**SMSQ AND ST/QL NOTES**

If BAUD is only followed by one parameter, then it sets the baud rates for both SER1 and SER2 on the QL, AURORA and QXL boards. However, if SMSQ/E is running on an ATARI computer, or the command is used on an ST/QL Emulator then it only sets the baud rate on SER1.

You can however supply two parameters to the command to set independent baud rates (note that on a standard QL or Aurora, Hermes is needed for independent baud rates on each serial port). In this case, the first parameter is the number of the serial port to be set and the second number is the new baud rate, for example:

BAUD 1,19200

sets the baud rate on SER1 to 19200 - any other serial ports are left unaffected. If the rate (bps) is specified as zero, this selects the highest possible BAUD rate on that port.

Please also note that if a translate has been set up with the TRA command, changing the BAUD rate will make that translate apply to all channels opened to the serial ports, whether or not they are already open. See TRA for more details.

The following additional BAUD rates are also supported on the specified SMSQ/E version:

**GOLD CARD & SUPER GOLD CARD**

- 1275(1200 receive and 75 transmit - only works with HERMES)
- 75(75 receive and 1200 transmit - only works with HERMES)

(The standard 1200 and 75 Baud rates are not supported)

**ATARI ST and TT**

On these computers, the different serial ports support different baud rates. An ST/STE only has one serial port (SER1), a Mega STE has three (SER1, SER2 and SER4), and a TT has four (SER1, SER2, SER3 and SER4).

Support for SER2, SER3 and SER4 was only added to the ST/QL Emulators in version E-37 of the Drivers. It has always existed in SMSQ/E.

SER1

- supports all the standard baud rates from 300 to 19200, except 7200.

SER2

- supports all the standard baud rates from 300 to 19200 (including 7200) as well as 38,400, 76,800, 83,333 and 125,000 baud (1x and 2x MIDI speeds).
  - If the rate specified is 0, the rate used is 153,600.
- Note that 38,400 on the TT was implemented in v2.69. 38,400, 76,800, 83,333, 125,000 and 153,600 BAUD were implemented for the STE and TT in v2.73.

SER3

- supports the same rates as SER1.
- Hardware handshaking is not available on this port.

SER4

- supports all the standard baud rates from 300 to 38,400 plus 57,600.
- If the rate specified is 0, the rate used is 230,000.

**QXL**

All of the standard baud rates available to the normal QL are supported except for 75 Baud.

**QPC**

All of the same baud rates as the QXL implementation are supported plus 38,400 and 57,600 baud.

**QXL AND QPC NOTES**

If one of the PC's serial ports is already linked to a mouse (in DOS) then the BAUD command will not affect that port.

**CROSS-REFERENCE**

The Devices Appendix supplies details about the serial device ser and parallel device par. The ser_… and par_… commands allow you to set various other parameters for serial and parallel ports.

You can check the current baud rate setting with *BAUDRATE*.

## 9.17 BAUDRATE

| | |
|---|---|
| Syntax | BAUDRATE |
| Location | SERMouse |

This function returns the actual baud rate of the system which will be used on any newly opened serial port channel.

**CROSS-REFERENCE**

The system's baud rate is set with *BAUD*.

## 9.18 BCLEAR

| | |
|---|---|
| Syntax | BCLEAR |
| Location | BeuleTools, TinyToolkit, BTool |

Each console channel has what is known as an input queue, a small area of memory where key presses are stored before they are read by INPUT, INKEY$ etc. The command BCLEAR clears the buffer of the current input queue so that any key presses which have not yet been processed are not seen by the program. This is useful to prevent overrun on keys.

**Examples**

(1) Type this line as a direct command into the interpreter, press <ENTER> and then type some keys. REPeat a: REMark

Now press break and all of those key presses which you performed after entering the line will be shown. Replace REMark by BCLEAR and try the same.

Normally it is okay for all key presses to be stored in a buffer - if a program cannot cope with the typist's speed, no key presses will be lost. But sometimes this feature may not be welcome.

(2) Even on very good keyboards the phenomenon of key-bounce appears, where the user has pressed a key once but the program thinks that the same key has been pressed a few times. This generally happens with poor quality keyboards or if the user is not used to either the keyboard or program speed.

This is a queue clearing version of the GETCHAR% function shown at CUR. Dangerous inputs should always clear the keyboard queue, for example where the program is asking the question: "Do you really want to format that disk (y/n) ?"

```
100 DEFine FuNction GETCHAR% (channel,timeout)
110  LOCal char$,dummy
120  dummy=PEND(#channel): BCLEAR
130  CUR #channel,1
140  char$=INKEY$(#channel,timeout)
150  CUR #channel,0
160  RETurn CODE(char$)
170 END DEFine GETCHAR%
```

**CROSS-REFERENCE**

The current keyboard queue can be selected by a dummy *INKEY$* or *PEND*.

## 9.19 BEEP

| | |
|---|---|
| Syntax | BEEP length, pitch [,pitch_2, grd_x, grd_y [,wrap [,fuzz [,rndom ]]]] or BEEP |
| Location | QL ROM |

This command allows you to access the QL's rather poor sound generation chip. It can be extremely difficult to use this command, and a lot of trial and error will generally be needed before you can find anything similar to the sound you are after.

BEEP without any parameters will turn off the sound altogether. You must also be aware of the fact that as soon as a BEEP command is encountered, the QL will cancel the current sound and emit the new one (whether or not the earlier sound had finished).

Each of the various parameters have different ranges and different effects on the sound produced:

- *length* This specifies the duration of the sound in 72 microsecond units (there are one million microseconds in a second). A length of zero means emit the sound until another BEEP command is encountered. The range is 0…32767 (a value of 32767 lasts for approximately 2.36 seconds).
- *pitch* This affects the tone of the sound produced. The allowable range is 0…255. A pitch of 0 is the highest which can be produced, ranging to 255 which is the deepest tone. The purity of the sound will be affected if any other parameters are specified.
- *pitch_2* This represents a second pitch level, which will have no effect if the tone is the same (or higher) than pitch. If however, the value of this parameter is higher (the tone is lower) than that of pitch, this specifies a range between which the sound can 'bounce' by use of the next two parameters, creating a sequence of notes (the length of the sequence will depend on the length parameter).
- *grd_x* Assuming that the BEEP command is now being used to produce a sequence of notes, this parameter specifies the time interval (in 72 microsecond units) of each note in the sequence. The permitted range is again 0…32767. Larger time intervals make each note in the sequence more distinct (low values tend to produce just buzzing).
- *grd_y* This parameter specifies the step between each note in the sequence. This must be in the range 0…15. However, this may make more sense if the correct range was said to be -7..8.

  A value of zero produces no step - you are returned to a single note again.

  A value between 1 and 7 means that each note will be that many pitches below the last one (unless that would bring the pitch below *pitch_2*).

  A value of 8 makes the BEEP command fit as many notes into the sequence (in the range) as possible.

  Values of 9 to 15 (or -7 to -1) mean that each note will be that many pitches above the last one (these correspond to the values 7 to 1 respectively), unless this would bring the pitch above pitch. When the top or bottom of the range pitch to pitch_2 is reached, the step direction is reversed to cause the sound to 'bounce'.
- *wrap* If this parameter is specified, the range of notes between the two pitch parameters will be repeated the specified number of times before the step direction is altered. The range for this parameter is 0..15.

  The last note in the range will not be sounded, but will appear as the first note in the opposite direction.
- *fuzz* This affects the purity of each note, by blurring its sound. The effective range is 8…15, with a value of 15 producing an awful buzz.
- *rndom* This parameter allows you to specify a certain amount of 'randomness' which is to be added to each note.

  The effective range is once again 8…15, with the given value being used to alter from how far away from the original sequence the QL can pick a note. The higher the value, the more random notes appear in the sequence.

**Examples**

BEEP 0,20,40,10070,2

will keep sounding every other note between 20 and 40 moving down and then up the scale.

BEEP 0,20,30,10070,2,1

will sound the notes in the following sequence 20, 22, 24, 26, 28, 20, 22, 24, 26, 28, 30, 28, 26, 24, 22, 30, 28, 26,. . . .

**NOTE 1**

On all ROMs if you set a very high pitch value, the QL finds it very difficult to read the keyboard. BEEP 0,0 and BEEP 0,1 will make typing rather difficult.

**NOTE 2**

Unless used on a THOR XVI, BEEP does not enter the QL into supervisor mode and therefore if BASIC is trying to use BEEP whilst a task is loaded or unloaded, then the system is likely to crash!

**NOTE 3**

BEEP does not do anything on ST/QLs or the Amiga-QDOS Emulator (pre v3.23).

**NOTE 4**

The pitch of the sound is actually shifted on QLs by different values of length, fuzz and rndom. The length of the sound is also somewhat dependent on the pitch! Both of these problems are however fixed by the replacement co-processor Hermes.

**CROSS-REFERENCE**

*BEEPING* allows you to check if a sound is currently being emitted.

*PAUSE* allows you to specify a time interval during which the computer will wait (allowing you to play much longer notes).

## 9.20 BEEPING

| | |
|---|---|
| Syntax | BEEPING |
| Location | QL ROM |

This is a simple function which returns either 1 (true) if any sound output from BEEP is still running or 0 (false) if not.

**Example**

BEEPING is rather useless in a formulation like: IF BEEPING THEN BEEP

because this is less efficient than BEEP on its own which has the same effect. However, where you want to ensure that your program generates the chosen sound, because of the QL's multi-tasking abilities, it may be useful to use this function in case another program is executing a BEEP command when you want to - you could then either wait or simply override that sound by using BEEP followed by your own sound generating BEEP command. For example:

```
10 REPeat check_beep: IF NOT BEEPING THEN EXIT check_beep
20 BEEP 100,20
```

**NOTE**

This function did not work correctly on Minerva before v1.98.

**CROSS-REFERENCE**

*BEEP* activates the speaker.

## 9.21 BELL

| | |
|---|---|
| Syntax | BELL |
| Location | ST/QL, QSound |

This command produces the sound of a ringing phone.

**CROSS-REFERENCE**

*SND_EXT*, *SHOOT*, *EXPLODE*.

## 9.22 Beule_EXT

| | |
|---|---|
| Syntax | Beule_EXT |
| Location | Beuletools |

This command is used to update all of the keywords which are added by the Beuletools Toolkit. The new keywords were automatically added when the Toolkit was loaded but keywords can be overwritten by other Toolkits, renamed or ZAPped.

Beule_EXT undoes these changes and restores the default status.

**WARNING**

Do not load the Beuletools toolkit into anything other than resident procedure memory (ie. do not have any Jobs running other than Job 0 when the toolkit is loaded). This may crash the system.

**CROSS-REFERENCE**

*TK2_EXT* and *TINY_EXT* do the same for Toolkit II and TinyToolkit keywords.

See also *ATARI_EXT*.

## 9.23 BGCOLOUR_QL

| | |
|---|---|
| Syntax | BGCOLOUR_QL [#ch,] colour |
| Location | SMSQ/E v2.98+ |

It is possible under the latest version of SMSQ/E to set a 'wallpaper' - this is an image which covers the whole of the available screen (in any resolution) and which forms a background for any programs which may be running. Normally, this would appear as a black area of the screen.

A valid window channel must be open, default #1 (or #0 on a SBASIC with only #0 open), although one may also be supplied as #ch. BGCOLOUR_QL allows you to specify any standard QL colour -

the parameters allowed are the same as for the INK command (in either Standard QL Colour Mode or COLOUR_QL mode), which thus allows for you to specify composite colours as well as palette mapped colours with PALETTE_QL.

**Example**

BGCOLOUR_QL 2,7 - sets a red and white checkerboard pattern.

**CROSS-REFERENCE**

Refer to Appendix 16 and *INK* for more details on colours.

*BGCOLOUR_24* is similar.

*BGIMAGE* may be used to set a screen image as the wallpaper.

## 9.24 BGCOLOUR_24

| | |
|---|---|
| Syntax | BGCOLOUR_24 [#ch,] colour |
| Location | SMSQ/E v2.98+ |

This is similar to BGCOLOUR_QL in that it allows you to set a wallpaper to cover the whole of the available screen (in any resolution).

A valid window channel must be open, default #1 (or #0 on a SBASIC with only #0 open), although one may also be supplied as #ch.

BGCOLOUR_24 allows you to specify any 24 Bit Colour - the parameters allowed are the same as for the INK command (in COLOUR_24 mode), which thus allows for you to specify composite colours.

**Example**

BGCOLOUR_24 $920000,$ff0092,3 -sets a checkerboard pattern of Dark Red and Shocking Pink.

**CROSS-REFERENCE**

Refer to Appendix 16 and *INK* for more details on colours.

*BGCOLOUR_QL* gives more detail.

*BGIMAGE* may be used to set a screen image as the wallpaper.

## 9.25 BGET

| | |
|---|---|
| Syntax | BGET [#ch\position,] [item $^{*}$[,item$^{i}$]$^{*}$ ..] or<br>BGET [#ch,] [item |
| Location | Toolkit II, THOR XVI |

This command is very similar to GET, although this only fetches one byte at a time (in the range 0..255) from the given channel (default #3).

Each item to be fetched must therefore be either an integer or a floating point variable, for example: BGET #3\100,byte1%,keying

If the channel specified is not a file, then the command will wait for a key to be pressed for each item, and then set the value of each item to the character code of the key pressed.

As with GET, the items will be fetched from the current (or specified) file position, which is taken to be an absolute distance from the start of the file. If no item is specified, then the first variant can be used to set the current file position. position will be updated (unless it is an expression!) with the current file position at the end of the command.

**Examples**

BGET #3\100 Set file pointer on #3 to position 100.

BGET a% Read the byte at the current file pointer in channel #3.

**NOTE 1**

Current versions of the Turbo and Supercharge compilers are not able to compile programs which use BGET.

**NOTE 2**

Characters which are read from a channel using BGET are affected by TRA.

**SMS NOTE**

BGET will accept a parameter which is a sub-string of a string array to read in several bytes at once. For example:

DIM a$(10):a$=FILL$(' ',10):BGET #3,a$(4 to 7)

This will read 4 bytes from channel #3 into the middle of a$.

Please note that a$ cannot be an empty string if this is to work since the sub-string would not be valid!!

**CROSS-REFERENCE**

See *BPUT*, *PUT*, *GET*. *FPOS* allows you to find out the current file position. *TRUNCATE* allows you to truncate a file to the current file position. *PEEK* fetches one byte from memory.

*OPEN_DIR* contains an example of the use of *BGET*.

## 9.26 BGIMAGE

| | |
|---|---|
| Syntax | BGIMAGE [#ch,] filename |
| Location | SMSQ/E v2.98+ |

This command allows you to load a screen image as a wallpaper to cover the whole of the available screen (in any resolution).

A valid window channel must be open, default #1 (or #0 on a SBASIC with only #0 open), although one may also be supplied as #ch.

The file will need to be a screen snapshot - the Photon JPEG viewer can be used to convert JPEG files to screen snapshots.

**Example**

BGIMAGE win1_wallpapers_cats - load a wallpaper.

**NOTE**

The command expects the screen to have been saved in the current resolution and colour depth, therefore any attempt to load a screen image in a different resolution or colour depth to the one in existence when the screen was saved will result in a corrupt image.

**CROSS-REFERENCE**

*SBYTES* gives details on how to store a screen on disk.

In many ways, this command is similar to *LOADPIC*, except that it caters for non-standard QL resolutions and colour depths.

*BGCOLOUR_QL* and *BGCOLOUR_24* can be used to set a single colour wallpaper.

## 9.27 BICOP

| | |
|---|---|
| Syntax | BICOP |
| Location | HCO |

This procedure will send a screen dump to the port ser1hr - it is aimed at Epson compatible dot-matrix printers and uses grey scales to represent the different colours. It is up to you to set the BAUD rate.

**NOTE**

It will only work on a standard 512x256 screen stored at $20000.

**CROSS-REFERENCE**

*SDUMP* is more flexible.

See also HCO and FCO.

## 9.28 BIN

| | |
|---|---|
| Syntax | BIN (binary$) or<br>BIN (binary) where binary=0..111111 |
| Location | Toolkit II, THOR XVI |

This function returns the decimal value of a binary number (given as a string). For small numbers, a floating point number can be used but will cause problems if this is not a valid binary number.

**Examples**

(1) PRINT BIN ( '1001' )

will print the value 9

(2) As it stands, the function BIN cannot handle binary dots

(eg. 1001.101=9.625), therefore a BASIC function to provide this facility is:

```
100 DEFine FuNction BINN(a$)
110   LOCal i,dot,dota,value_a,loop
120   IF a$='' THEN RETurn 0
130   FOR i=1 TO LEN(a$): IF a$(i) INSTR '10.'=0: REPORT -17: STOP
```

(continues on next page)

(continued from previous page)

```
140   dot='.' INSTR a$: IF dot=0 THEN RETurn BIN(a$)
150   value_a=0:dota=0
160   IF dot>1 THEN value_a=value_a+BIN(a$(1 TO dot-1))
170   IF '.' INSTR a$(dot+1 TO ): REPORT -17: STOP
180   REPeat loop
190     IF dot>=LEN(a$):EXIT loop
200     a$=a$(dot+1 TO )
210     dot='1' INSTR a$: IF NOT dot THEN EXIT loop
220     value_a=value_a+1/(2^(dot+dota)):dota=dota+dot
230   END REPeat loop
240   RETurn value_a
250 END DEFine BINN
```

**NOTE**

Any digit other than 0 or 1 will produce odd results.

**CROSS-REFERENCE**

*BIN$* works the other way around, converting decimal numbers into their binary equivalent. See *HEX* and *HEX$* for the hexadecimal versions. *BIT%* is also useful.

SMS users can achieve the same thing by using, for example *PRINT*%1001 instead of *PRINT*BIN('1001').

## 9.29 BIN$

| | |
|---|---|
| Syntax | BIN$ (decimal,digits) or<br>BIN$ (decimal [,digits]) (THOR only) |
| Location | Toolkit II, THOR XVI |

This function converts a signed integer decimal number to the equivalent binary number (to a specified number of binary digits ranging from 1 to 32). Negative values are also handled correctly.

**Examples**

(1) BIN (BIN$ (x,4)) = x

if x is any number between 0 and 15.

(2) A short function to compare two numbers using the mathematical 'OR' function. Do note however that the same function already exists on the QL, and the commands

PRINT 100||10 and PRINT _or(100,10) have exactly the same effect, although the BASIC version below does enable you to see what the function actually does:

```
100 DEFine FuNction _or(x,y)
110  a$=BIN$(x,32): b$=BIN$(y,32)
115  PRINT a$,b$
120  c$=FILL$('0',32)
130  FOR i=1 TO 32
140    IF a$(i)=1 OR b$(i)=1: c$(i)=1
```

(continues on next page)

(continued from previous page)

```
150  END FOR i
155  PRINT c$
160  RETurn BIN(c$)
170 END DEFine _or
```

**THOR XVI NOTE**

The THOR XVI version of BIN$ will accept a value of zero for digits {or even the command in the form BIN$(decimal)}. In both of these cases the result is returned in the least number of Binary digits necessary to store the number, for example: PRINT BIN$(10)gives the result 1010.

**THOR XVI WARNING**

A second parameter of zero may crash some versions of this command other than on v6.41 of the THOR XVI.

**CROSS-REFERENCE**

See *BIN* and *HEX*, *HEX$*. Also refer to *BIT%*.

## 9.30 BINOM

| | |
|---|---|
| Syntax | BINOM (n,k) |
| Location | Math Package |

The function BINOM returns the value of the binomial coefficient which is defined as the following (where n >= k):

n * (n - 1) * (n - 2) * … * (n - k + 1) ) / ( 1 * 2 * … * k ) or, FACT(n) / ( FACT(k) * FACT(n-k) )

The binomial coefficient gives the kth coefficient of the variables in an expanded binomial series, this is called the binomial theorem:

(a+b)^n = BINOM(n,0) * a^n + BINOM(n,1) * a^(n-1) * b + BINOM(n,2) * a^(n-2) * b^2 + … + BINOM(n,n-1) * a * b^(n-1) + BINOM(n,n) * b^n

The binomial coefficient can also be used to calculate combinations and probabilities. As the example shows, it is important to know the mathematical theory behind this function to make full use of it.

**Example**

The following program calculates $2^n$:

```
100 n=10: s=0
110 FOR k=0 TO n: s=s+BINOM(n,k)
120 PRINT s,2^n
```

It can be optimised by exploiting the fact that:

BINOM (n,k) = BINOM (n,n-k) which saves half of the loops:

```
100 n=10
110 IF NOT n MOD 2 THEN s=BINOM(n,n DIV 2): ELSE s=0
120 FOR k=0 TO n DIV 2 - NOT n MOD 2
```

(continues on next page)

(continued from previous page)

```
130   s=s+2*BINOM(n,k)
140 END FOR k
150 PRINT s,2^n
```

**CROSS-REFERENCE**

*FACT*.

## 9.31 BIT%

| | |
|---|---|
| Syntax | BIT% (number%,bitnr) with bitnr=0..15 |
| Location | BIT |

All numbers are internally stored as a series of values, each of which can either be 1 or 0 (or, if you prefer, true or false). This is known as the binary system. The set of digits which make up a binary number are known as a stream of bits.

The function BIT% returns the status of a specified bit of an integer number%, a value of either 0 or 1. Bit 0 means the rightmost bit, whereas bit 15 would be the leftmost.

**Example 1**

Here is a function which converts a number to the binary system. It allows a greater range than BIN$ and needs just one parameter. The first version needs the REV$ and LOG2 extensions, the second does not.

Version 1:

```
100 DEFine FuNction BIT$ (x%)
110  LOCal b$,i: b$=""
120  FOR i=0 TO LOG2(ABS(x%)):b$=b$ & BIT%(x%,i)
130  RETurn REV$(b$)
140 END DEFine BIT$
```

Version 2:

```
100 DEFine FuNction BIT$ (x%)
110  LOCal b$,i: b$=""
120  FOR i=0 TO LN(ABS(x%))/LN(2): b$=BIT%(x%,i) & b$
130  RETurn b$
140 END DEFine BIT$
```

**Example 2**

The following logical function returns 1 (true) if the given parameter was an upper case character, or 0 (false) if it was lower case. This function will work with all international character sets supported on the original QL.

```
100 DEFine FuNction UPPER% (c$)
110  RETurn NOT BIT%(CODE(c$),5) ^^ BIT%(CODE(c$),7)
120 END DEFine UPPER%
```

In any given character, bit 5 indicates the case and bit 7 the character set.

**CROSS-REFERENCE**

*BIN$* also converts a decimal number to a binary and *BIN* back again. *UPPER$* returns a string in upper characters.

The length of a number x in binary form is *INT* (LOG2 (ABS (x))+1) .

## 9.32 BLD

| | |
|---|---|
| Syntax | BLD |
| Location | Beuletools |

This function returns the control codes needed to switch on double strike ('bold') on an EPSON compatible printer:

PRINT BLD is the same as PRINT CHR$(27)&"G"

**Example**

LPRINT "I " & BLD&"hate"&NRM & " these functions."

**CROSS-REFERENCE**

*NORM*, *EL*, *DBL*, *ENL*, *PRO*, *SI*, *NRM*, *UNL*, *ALT*, *ESC*, *FF*, *LMAR*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 9.33 BLOCK

| | |
|---|---|
| Syntax | BLOCK [#channel,] width, height, x, y, colour |
| Location | QL ROM |

This command draws a block of size width x height at position x,y of the given colour in the specified window (default #1). Both the position and the block size are given in absolute pixel co-ordinates, with the maximum ranges specified by the physical size of the window.

This means that for example, in a window which is defined as 448x200a32x16, the maximum block which can be drawn is a block of size 448 x 200 in position (0,0). You can also use OVER to produce other effects with BLOCK.

As with other graphics commands, the colour can be made up of up to three parameters, giving the background, contrast and stipple pattern (composite colours).

**Example**

A program printing out the set of numbers 1 to 100 and then quickly recolouring the two halves of the window:

```
100 WINDOW 300,60,102,56
110 PAPER 0: CLS
120 FOR i=1 TO 100: PRINT !i!
130 OVER -1
140 BLOCK 150,60,0,0,7
150 BLOCK 150,60,150,0,2
160 OVER 0
```

**NOTE 1**

Some ROMs (not SMS) will allow you to specify blocks which lie partly outside of a window without reporting the error 'Out of Range'. However, this can also crash some ROMs!

**NOTE 2**

Odd values for width and x are always rounded down to an even number (eg. 23 pixels wide becomes 22 pixels). This is to ensure compatibility between MODE 4 and MODE 8. The only problem is that you cannot specify a block one pixel wide, or even have a gap of one pixel between two blocks.

**NOTE 3**

Unless you have a Minerva ROM or SMS, you cannot draw a block 512 pixels wide - you need to use two adjacent blocks instead!

**NOTE 4**

BLOCK provides an extremely quick way of drawing horizontal or vertical lines on screen.

**MINERVA NOTE**

Early versions of Minerva (pre v1.83) contained code to ensure that the given block would appear wholly within the specified window. However, later versions had to be altered to ensure compatibility with certain programs. These later versions allow width, height, x and y to be within the range -32768…32767 - only that part of the block (if any) which appears in the given window will be drawn!

For example:

BLOCK 200,10,-20,255,7 has the same effect as:

BLOCK 180,1,0,255,7

**CROSS-REFERENCE**

*INK* contains information concerning composite colours.

## 9.34 BLOOK

| | |
|---|---|
| Syntax | BLOOK (tofind$, adr1 TO adr2) |
| Location | HCO |

See SEARCH but note the different syntax. The string being looked for by BLOOK is not case-sensitive.

## 9.35 BLS

| | |
|---|---|
| Syntax | BLS time% |
| Location | SERMouse |

This command sets up a job which will blank the screen after a certain amount of time if a key is not pressed or the mouse not moved. The delay depends on the value of time% (1-20 = seconds), (21-59 = minutes).

To turn off this function, use time%=0.

Pressing a key or moving the mouse will reactivate the screen.

**CROSS-REFERENCE**

See *A_BLANK*.

## 9.36 BMOVE

| | |
|---|---|
| Syntax | BMOVE adr1a, adr1b, adr2 |
| Location | HCO |

BMOVE is a procedure which copies the whole of the memory stored between the two addresses adr1a and adr1b to the new address pointed to by adr2, so the number of bytes moved is adr1b-adr1a.

**Example**

Dump some memory to screen (note that this only works with the screen at 131072 and at 512 x 256 resolution):

```
100 FOR a = 0 TO 10240 STEP 128
110   BMOVE a, a+HEX("8000") TO HEX("20000")
120 END FOR a
```

**WARNING**

Always ensure that there is sufficient available free memory at adr2 to hold the data from adr1a to adr1b, otherwise your machine is most likely to crash.

**CROSS-REFERENCE**

It is a good idea to check with *VER$* if Minerva is present and use its extremely fast MM.MOVE machine code trap via *CALL* as an alternative to *BMOVE*; this is demonstrated by the example at *LDRAW*.

*COPY_B*, *TTPOKEM*, *COPY_W* and *COPY_L* also allow you to move memory.

## 9.37 BORDER

| | |
|---|---|
| Syntax | BORDER [#channel,] size [,colour] or<br>BORDER [#channel] |
| Location | QL ROM |

This command allows you to add a coloured border around the inside of the edge of the specified window (default #1). If the second syntax is used, this will turn off the border on the specified window - this is the same as:

BORDER [#channel,] 0

If a border is present around the window, the physical size of the window is altered, so that PRINT and LINE commands (for instance) will not overwrite the border. Please note however, that the window is reset to its original size prior to a BORDER command and therefore two successive border commands only have the same effect as the second BORDER command on its own.

If the specified size is too large to fit in the given window, the error 'Out of Range' will be reported.

As with other graphics commands, colour can actually be up to three parameters forming a composite colour.

For example: BORDER #2,2,4

has the same effect as BORDER #2,2,4,4,3

or even BORDER #2,2,4,4

If no value is given for colour a transparent border will be added to the given window. This means that a border will be created, but anything which already appears in that border will not be affected.

Once the border has been re-drawn the cursor is automatically placed at the top left hand position (0,0) just inside the border.

**Examples**

To produce a screen with a title, allowing you to scroll

text and do all sorts on the screen without affecting the title:

```
100 MODE 4
110 WINDOW 448,200,32,16
120 PAPER 0: BORDER 0 : CLS
130 AT 0,30: PRINT "THE TITLE PAGE"
140 BORDER 9
150 PAPER 2:CLS
```

To produce a 'take-off' effect:

```
100 MODE 8
110 WINDOW 448,200,32,16
120 FOR i=1 TO 99
130   BORDER i,2
140 END FOR i
```

**NOTE 1**

If a border appears in a window, there is always a width of at least two pixels down the sides to ensure that the border will appear in MODE 8. Take the width value and if it is odd, add one for the width down the sides of the window.

**NOTE 2**

The second syntax will not work on Minerva (pre v1.79) and the THOR XVI - you will need to specify a width of zero instead.

**CROSS-REFERENCE**

*INK* describes composite colours.

Also see *WINDOW*.

## 9.38 BPEEK%

See *BPEEK_L* below.

## 9.39 BPEEK_W%

See *BPEEK_L* below.

## 9.40 BPEEK_L

| | |
|---|---|
| Syntax | BPEEK% (offset) and<br>BPEEK_W% (offset) and<br>BPEEK_L (offset) |
| Location | BPEEKS, BPOKE (DIY Toolkit - Vol B) |

These three functions are exactly the same as BASIC_B, BASIC_W and BASIC_W, and suffer with the same problem that they always access the SuperBASIC variables of Job 0 (SuperBASIC) and cannot therefore be used on a Multiple BASIC interpreter.

**CROSS-REFERENCE**

See *BASIC_W* and *BASIC*.

*BPOKE* and related commands allow you to alter the values of the SuperBASIC variables.

## 9.41 BPOKE

See *BPOKE_L* below.

## 9.42 BPOKE_W

See *BPOKE_L* below.

## 9.43 BPOKE_L

| | |
|---|---|
| Syntax | BPOKE offset, value and<br>BPOKE_W offset, value<br>BPOKE_L offset, value |
| Location | BPOKE (DIY Toolkit - Vol B) |

These three commands allow you to alter the value of SuperBASIC variables in much the same was as the extended POKE commands do on Minerva and SMS.

BPOKE_W and BPOKE_L were added in v0.7 of the toolkit.

They unfortunately always access the SuperBASIC variables of Job 0 (SuperBASIC) and cannot therefore be used on a Multiple BASIC interpreter.

**CROSS-REFERENCE**

See *POKE*.

*BPEEK%* and related commands allow you to read the values of the SuperBASIC variables.

## 9.44 BPUT

| | |
|---|---|
| Syntax | BPUT [#ch\position,] [item *[,item$^{i}$]* ..] or<br>BPUT [#ch,] [item |
| Location | Toolkit II, THOR XVI |

This command is the complement to BGET, in that it places the byte value for each item into the specified channel (default #3) at the current file position (or the specified position if the first variant is used).

If the value of item is outside the range 0…255, an overflow error will result, whereas if item is not an integer or floating point number, then an error in expression will result. On the other hand, if a non-integer floating point is given as an item, then the value will be rounded to the nearest integer and this placed into the given channel.

Provided that the second variant of this command is used, the specified channel need not be open to a file, in which case each item is taken as being a character, for example: BPUT #2,72,101,108,108,111

will print the word Hello in channel #2. This can of course be used to send control codes to a printer much more easily than the PRINT command.

For example:

BPUT #3,27,70

is a lot easier to understand than:

PRINT #3,CHR$(27)&'F'

to switch off emphasised mode.

As with BGET, if no item is specified, then the first variant can be used to set the current file position. position will also be updated at the end of the command to contain the current file pointer.

**Example**

BPUT #ch,4.5,'100',52,a+1

places the values 5,100,52 and (a+1) at the current file position.

**NOTE**

The codes sent by BPUT are affected by any translate that is active (see TRA).

**SMS NOTE**

BPUT will now accept string parameters to allow you to pass several bytes at a time, for example: a$='Hello':BPUT #3,a$

is equivalent to: BPUT #2,72,101,108,108,111

**CROSS-REFERENCE**

See *FGETB*, *BPUT*, *PUT*, *GET*, *LPUT*, *UPUT* and *WPUT*.

*FPOS* allows you to find the current file position.

*TRUNCATE* allows you to truncate a file to the current file position.

*PEEK* fetches one byte from memory.

*UPUT* allows you to send bytes without them being translated.

## 9.45 BREAK_ON

See *BREAK_OFF* below.

## 9.46 BREAK_OFF

| | |
|---|---|
| Syntax | BREAK_ON<br>BREAK_OFF |
| Location | TinyToolkit |

The command BREAK_OFF de-activates the functioning of both <CTRL><SPACE> (the Break Key) and <CTRL><F5> (the Pause Screen key) during the running of interpreted SuperBASIC programs so that they cannot be stopped by the user unless they stop either due to an error or a STOP command.

The command BREAK_ON reactivates both keys.

The function BREAK returns the current status:

IF BREAK=1 means the Break Key is active, while

IF BREAK=0 means that it is inactive.

**NOTE 1**

BREAK_OFF may not work on Minerva ROMs unless you have v1.10 or later of the Toolkit, which uses the new Minerva System Xtensions to overcome any problem.

**NOTE 2**

BREAK_OFF does not currently work with SMS.

**CROSS-REFERENCE**

*STOP* terminates interpreted programs even if the Break Key is disabled. Do not confuse with the command *BREAK*.

## 9.47 BREAK

| | |
|---|---|
| Syntax | BREAK switch |
| Location | BTool |

The command BREAK takes the parameter of either ON (=1) or OFF (=0) and enables or disables the ability to stop a program with the Break key <CTRL><SPACE> (and <ESC> on Minerva) accordingly.

**Example**

```
100 WINDOW 136,100,100,40: INK 7
110 BORDER 1,4,3: PAPER 3,0: CLS
120 SCALE 100,-50,-50: POINT 0,0
130 fast=ASK( "Fast (y/n)" ): CLS
140 BREAK fast
150 FOR n=0 TO 4000
160   IF BREAK% THEN AT 0,0: PRINT n
```

(continues on next page)

(continued from previous page)

```
170   x=RND(-50 TO 50): y=RND(-50 TO 50)
180   z=SIN(PI*SQRT(x*x+y*y)/10)+1
190   IF z > 2*RND THEN POINT x,y
200 END FOR n
210 BREAK ON
```

**NOTE 1**

After the Break key has been disabled and re-enabled, if you try to Break from the interpreter's command window #0 it might be disturbed. Instead of printing 'not complete' (error -1) in #0 when <CTRL><SPACE> is pressed, that message may appear in #2 and Break will work only once, the interpreter will not accept any further Breaks… A single <ENTER> after you initially press the Break key cures this.

**NOTE 2**

This command does not work under SMS.

**CROSS-REFERENCE**

See also *BREAK%*, *FREEZE* and *FREEZE%*.

Do not confuse BTool's command *BREAK* with TinyToolkit's function *BREAK* (although you can use both in the same program!)

## 9.48 BREAK%

| | |
|---|---|
| Syntax | BREAK% |
| Location | BTool |

The function BREAK% returns the current state as to whether the Break key is enabled, either ON or OFF.

**CROSS-REFERENCE**

See *BREAK*!!

## 9.49 BTool_EXT

| | |
|---|---|
| Syntax | BTool_EXT |
| Location | BTool |

This command is similar to TK2_EXT and TINY_EXT, in that it installs BTool so that keyword definitions with the same name as those provided in other Toolkits are overwritten with the Btool definition.

**WARNING**

BTool_EXT will hang SuperBASIC if the BTool Toolkit has been loaded into the common heap - this is most likely to happen on later versions of Toolkit II where LRESPR uses the common heap if jobs are running. Try LINKUP instead.

See also *KILL* which removes all current jobs.

## 9.50 BTool_RMV

| | |
|---|---|
| Syntax | BTool_RMV |
| Location | BTool |

All keywords implemented by BTool (except BTool_EXT) are removed from the SuperBASIC name list. The Toolkit itself remains in memory and can be re-activated with BTool_EXT.

## 9.51 BTRAP

| | |
|---|---|
| Syntax | BTRAP #ch,key [,d1 [,d2 [,d3 [,a1 [,a2 ]]]]] |
| Location | TRAPS (DIY Toolkit Vol T) |

This command is identical to QTRAP, except that the address parameters (a1 and a2) are taken to be relative to A6, therefore allowing you to access system calls which need to access the SuperBASIC variables, so that you can for example save and load arrays direct!!

**WARNING**

Several TRAP #3 calls can crash the computer - make certain that you know what you are doing!

**CROSS-REFERENCE**

See *IO_TRAP*, *MTRAP* and *QTRAP*.

Any return parameters can be read with *DATAREG* and *ADDREG*.

*CLS*, *PAN* and *SCROLL* can also be used to call TRAP #3.

Refer to the QDOS/SMS Reference Manual (Section 15) for details of the various system TRAP #3 calls.

Also refer to the DIY Toolkit documentation for this command.

## 9.52 BUTTON%

| | |
|---|---|
| Syntax | BUTTON% (flag) |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I), Amiga QDOS v3.20+ |

This function can be used to find out if any mouse buttons have been pressed and if so which ones. Unfortunately, you cannot use this to find out if a button has been pressed twice quickly in succession (known as double-clicking).

The value of flag is used to tell the function which buttons you wish to interrogate:

- 0 - Has any key been pressed ? If so, the value returned will be 0 plus the following numbers if the relevant key(s) has been pressed:
  - +1 - Button One Pressed
  - +2 - Button Two Pressed
  - +4 - Button Three Pressed

- 1 - Has Button One been pressed (this is the left hand mouse button)? If so 1 is returned, otherwise 0.
- 2 - Has Button Two been pressed (this is the right hand mouse button)? If so, 1 is returned, otherwise 0.
- 3 - Has Button Three been pressed (this is the middle mouse button)? If so, 1 is returned, otherwise 0.

**Example**

A routine to wait for the user to press the right and left mouse button at the same time:

```
100 DEFine PROCedure WAIT_MOUSE
110 REPeat mloop
120   IF BUTTON%(0)=1+2:RETurn
130 END REPeat mloop
140 END DEFine
```

**CROSS-REFERENCE**

*X_PTR%*, *Y_PTR%* and *PTR_FN%* can also be used to interrogate the mouse.

## 9.53 BVER$

| | |
|---|---|
| Syntax | BVER$ |
| Location | BeuleTools |

This function returns the version number of the Beule Toolkit. This may be useful if a program makes use of commands or functions which were not supported by older versions.

## 9.54 BYTES_FREE

| | |
|---|---|
| Syntax | memory = BYTESFREE |
| Location | DJToolkit 1.16 |

This simple function returns the amount of memory known by the system to be free. The answer is returned in bytes, see also *KBYTES_FREE*. For the technically minded, the free memory is considered to be that between the addresses held in the system variables SV_FREE and SV_BASIC.

**EXAMPLE**

```
...
2500 freeMemory = BYTES_FREE
2510 IF freeMemory < 32 * 1024 THEN
2520    REMark Do something here if not enough memory left...
2530 END IF
...
```

**CROSS-REFERENCE**

*KBYTES_FREE*.

CHAPTER

TEN

# KEYWORDS C

## 10.1 CACHE_ON

| | |
|---|---|
| Syntax | CACHE_ON |
| Location | SMS, Super Gold Card |

This command enables any internal caches which may be available on your operating system. This is in fact the default.

Caches are a means of storing computer instructions in fast memory and cutting down on the time taken by a computer to execute those instructions.

Normally a computer chip works is fed a program in a series of numbers representing commands, a format which is known as machine code. This machine code operates at a very low level - the SuperBASIC command PRINT a$ would need several hundred machine code commands to have any effect on screen). The later Motorola chips (not 68000 or 68008) used by QLs and Amigas (and also the newer chips on PCs and ATARIs) all have on-board caches which can hold a certain number of these machine code instructions. If, while the program is running, it accesses those instructions again within a short time (ie. before the cache becomes full), then the chip can execute that series of commands again very quickly.

Although caches can therefore speed up many programs, some computer programs were written in the days before caches were available for the QL and compatibles, and therefore will not work if the cache is switched on. This is particularly true of some of the commands used by the Turbo compiler which contain self-modifying code, thus meaning that storage of a chunk of instructions is self-defeating.

**CROSS-REFERENCE**

*CACHE_OFF* allows you to disable the caches.

## 10.2 CACHE_OFF

| | |
|---|---|
| Syntax | CACHE_OFF |
| Location | SMS, Super Gold Card |

This command disables the internal caches used to speed up the operating system. This can help some of the older software to work on newer systems.

In particular, this command is needed if Flexynet is to work (see NETSEND).

**CROSS-REFERENCE**

See *CACHE_ON*.

## 10.3 CALL

| | |
|---|---|
| Syntax | CALL address [,d1[,d2[,d3[,d4[,d5[,d6[,d7 [,a0[,a1[,a2[,a3[,a4[,a5 ]]]]]]]]]]]]] |
| Location | QL ROM |

This command allows you to call a machine code routine loaded at the memory location address from SuperBASIC. At the same time, you can set the initial 68008 registers by supplying more than one parameter. Each additional parameter should be an integer value and is placed into the appropriate machine code register.

You cannot return values to SuperBASIC using this command, although you can return errors by setting D0 from the machine code on return.

If the machine code register D0 contains a number other than an error code (or 0) on return, the program will stop with the error 'At line '.

Various useful routines can be CALLed on a Minerva ROM - these are discussed on the next few pages.

**NOTE 1**

This command could crash the computer if used from within a program longer than 32K on pre JS ROMs. This is fixed by Toolkit II, the THOR XVI and Minerva.

**NOTE 2**

It can be dangerous to CALL addresses in memory unless you know that you have loaded a specified machine code routine into that location.

MINERVA NOTES:

**MINERVA NOTES**

Minerva adds various routines which can be CALLed from within a SuperBASIC program to perform various tasks quickly and efficiently. The routines are as follows:

**(1) Reset machine**

CALL 390,param

This routine resets the QL and allows you to set various parameters according to the value of param, which allows you, for example, to cut the amount of memory available to the machine. >/p>

To calculate the value of param, look at the following table and decide what effects you want the reset to have.

Next, look up the value of that effect and add it to param.

**EffectValue to add**

- Skip memory test = 1
- Skip ROM scanning (ignore extras) = 2
- Alter maximum memory = 4
- Default to TV mode = 8
- Ignore F1..F4 (no wait) = 16

- Leave n*256 bytes between screen and System Variables = n*256
- Set ramtop to nKB = (n+128)*1024

**Examples**

(a) reset machine and force dual screen TV mode (no memory test):

CALL 390,8+16+128+1

(b) reset machine to 640K:

CALL 390,(640+128)*1024+4

**MINERVA WARNINGS**

CALL 390 can crash the machine - always set the new ramtop to a multiple of 64K and do not try to allocate more memory than is in the system.

If you leave space between the screen and System Variables, this will reduce the amount of memory available accordingly!!

If your system uses a keyboard linked to SuperHermes, do not use CALL to reset the system unless you include a line such as PAUSE 40 prior to the CALL command to clear all pending input, otherwise SuperHermes becomes confused!

**(2) Move memory quickly:**

CALL PEEK_W(344)+16384,length,2,3,4,5,6,7,dest,source

This command allows you to move length bytes from the source address to the destination address (dest) extremely quickly.

Either source or dest may be odd addresses, and the code will even cope with overlapping areas.

Minerva Example:

**Minerva Example**

To copy the whole of the main screen to a screen storage area pointed to by the variable scr_store

```
10 scr_size=SCR_LLEN*SCR_YLIM
20 scr_store=ALCHP(scr_size)
30 CALL PEEK_W(344)+16384,scr_size,2,3,4,5,6,7,scr_store,SCR_BASE
```

**(3) Clear memory quickly**

CALL PEEK_W(360)+16384,length,2,3,4,5,6,7,address

This command allows you to clear length bytes from the given start address onwards extremely quickly. It could for example, be used to clear storage buffers.

Please note that address may be odd.

**CROSS-REFERENCE**

*LBYTES*, *SBYTES* can be used to load and save areas of memory (and machine code routines).

*ALCHP* and *RESPR* can be used to set aside areas of memory for user routines.

*BMOVE* and similar commands allow you to move areas of memory on other ROM implementations.

## 10.4 CAPS

| | |
|---|---|
| Syntax | CAPS |
| Location | BeuleTools |

After the command CAPS has been issued, any input from the keyboard via INPUT, INKEY$ etc. is translated into capital letters. CAPS simulates the use of the capslock key.

**Example**

To ask the user for any keyboard input, for example a

password where this should be entered in capital letters:

```
100 CAPS
110 INPUT "Please enter password:"!pass$
120 NOCAPS
```

**NOTE**

Some old replacement keyboards use dirty tricks to engage capslock. If you are fed up with the original QL keyboard then ensure you get the latest release of a modern keyboard interface and an IBM-style keyboard. If you do not do this, then you may have to change to capslock mode with CAPS…

**CROSS-REFERENCE**

*NOCAPS* is self-explanatory.

## 10.5 CARD_CREATE

| | |
|---|---|
| Syntax | CARD_CREATE card,size,filename$ |
| Location | SMSQ/E for Q68 |

Creates a new file on an SD card that is size MB in size with the name given by $filename.

Being a complex comand, it can return many different errors depending on what has gone wrong.

| Error | Num | Description |
|---|---|---|
| err.drfl | -1 | **Drive full**<br>there is no space for a file of this size within contiguous sectors on the card. |
| err.imem | -3 | **Insufficient memory for operation**<br>you shouldn't actuall get this error |
| err.orng | -4 | **File size out of range**<br>the projected size of the file is too big, or 0 or negative. |
| err.bffl | -5 | **Buffer full**<br>there is no space in the first 16 directory entries for a new file. |
| err.fex | -8 | **File already exists**<br>a file with this name already exists in the first 16 directory entries for a new file |
| err.inam | -12 | **Invalid file name**<br>not an 8.3 name. |
| err.ipar | -15 | **Wrong card number**<br>not 1 or 2. |
| err.mchk | -16 | **Medium check failed**<br>card wasn't readable (perhaps absent / not initialised) or this isn't a valid FAT32 partition. |

**Example**

```
10 CARD_CREATE 1,200,"test.win" : REMark create empty file, 200 MB in size
20 WIN_DRIVE 4,1,"test.win"     : REMark point win4_ to it
30 WIN_FORMAT 4                 : REMark allow formatting of win4_
40 FORMAT win4_name             : REMark format win4_
```

**Note 1**

There must be an empty slot in the first 16 entries of the FAT32 volume and the file name must obey the "8.3 rule".

**Note 2**

The file is not zeroed out and may contain random data.

**Note 3**

The maximum file size is dependant on the cluster size for the FAT32 volume.

| Cluster size | Max file size |
|---|---|
| 2 | just under 8 GB |
| 4 | just under 16 GB |
| 8 | just under 32 GB |

**Note 4**

SMSQ/E only checks the first 16 entries. If there are more than 16 and an empty entry exists in the first 16, if may be possible to create an duplicate entry with the name of an existing file.

**CROSS-REFERENCE**

*CARD_INIT*, *CARD_DIR$*

## 10.6 CARD_DIR$

| | |
|---|---|
| Syntax | CARD_DIR$(card) |
| Location | SMSQ/E for Q68 |

Returns a string containing the first 16 8.3 entries on the FAT23 partition of card 1 or 2.

Entries are separated by linefeeds (ascii 10) with "– Empty –" signifying an empty slot and "-Long name-" signifying a filename that does not fit the 8.3 format.

**CROSS-REFERENCE**

*CARD_CREATE*

## 10.7 CARD_INIT

| | |
|---|---|
| Syntax | CARD_INIT card_number |
| Location | SMSQ/E for Q68 |

This command initializes an SD card reader. By default only card 1 is active on boot.

**CROSS-REFERENCE**

## 10.8 CARD_RENF

| | |
|---|---|
| Syntax | CARD_RENF card,old$,new$ |
| Location | SMSQ/E for Q68 |

Renames a file in the first 16 entries of the FAT32 partition of an SD card.

## 10.9 CATNAP

| | |
|---|---|
| Syntax | CATNAP |
| Location | Turbo Toolkit |

The Turbo compiler allows PROCedure and FuNction definitions within a compiled program to be defined as GLOBAL and then called by other tasks. This is similar to making a modular machine code program which is then linked together when the assembly language modules are assembled. Under Turbo, the various program modules can be compiled separately, but then loaded together with LINK_LOAD_A and similar commands.

The CATNAP command will force a compiled program to wait at this statement indefinitely. The compiled program is only allowed to carry on execution from the next statement if another module calls one of the GLOBAL definitions contained in the current program and the GLOBAL PROCedure or FuNction has completed.

If CATNAP is used within a SuperBASIC program, then the program is simply suspended until the Break key is pressed.

**CROSS-REFERENCE**

SNOOZE is similar. See also GLOBAL, EXTERNAL and LINK_LOAD_A.

## 10.10 CBASE

| | |
|---|---|
| Syntax | CBASE [(#ch)](Btool) and<br>CBASE (#ch)(TinyToolkit) |
| Location | BTool, TinyToolkit |

The function CBASE finds the start address of the channel definition block which belongs to #ch. This is an area in memory where QDOS stores a lot of information about the channel, for example, which kind of device is connected to the channel.

The Btool variant returns the base of channel #1 if #ch is not specified.

**CROSS-REFERENCE**

The Pointer Interface modifies the structure of channel definition blocks for windows.

If you want to access these, preferably use WINCTRL instead of *CBASE*. See also *CHBASE*.

You can also use the *CHAN_ <KeywordsC.clean.html#chan_>*xx functions to look at the channel definition block.

## 10.11 CCHR$

| | |
|---|---|
| Syntax | CCHR$ (x) |
| Location | BTool |

The function CCHR$ takes a word value (max 32767) and returns two characters represented by that word. This is therefore the same as:

X=PEEK_W(10000)

PRINT CHR$(X DIV 256);CHR$(X MOD 256)

**CROSS-REFERENCE**

*CHR$* can be used to print each character separately.

## 10.12 CDEC$

| | |
|---|---|
| Syntax | CDEC$ (value,length,ndp) |
| Location | Toolkit II, THOR XVI |

The function CDEC$ allows you to convert a given value into a string in a specified format. This function will always take the integer part of the given value (which must be in the range -2^31…2^31, and will be rounded to the nearest integer if it is a floating point) and then assumes that the last ndp digits are to the right of the decimal point.

If there are enough characters to the left of the decimal point, a comma (',') will be placed between each set of three characters. The length is the length of the string which is to be returned, which must always be greater than or equal to the length of the value plus each comma and the decimal point. If length is not large enough, then the string returned will be full of asterisks ('*').

This function is particularly useful for formatting columns of figures, especially in view of the fact that it sidesteps the QL's habit of converting large numbers to exponential form. The commas ensure that it is ideal for use in formatting output of currencies.

**Examples**

PRINT CDEC$(123,4,0)

will print ' 123'

PRINT CDEC$(123,4,1)

will print '12.3'

PRINT CDEC$(1234567,9,2)

will print '12,345.67'

**CROSS-REFERENCE**

*PRINT_USING* is a general means of formatting output.

*IDEC$* and *FDEC$* are complementary functions.

## 10.13 CD_ALLTIME

| | |
|---|---|
| Syntax | CD_ALLTIME |
| Location | SMSQ/E for QPC |

This function returns the actual elapsed time in REDBOOK format from the start of the CD which is being played at present.

**Example**

A procedure to give the currently elapsed time:

```
100 DEFine PROCedure SHOW_TIME
110 elapse%=CD_ALLTIME
120 PRINT 'TOTAL ELAPSED TIME: ';CD_HOUR (elapse%);' HRS ';CD_MINUTE (elapse
↪%);' MINS ';:
130 PRINT CD_SECOND (elapse%);' SECS'
130 END DEFine
```

**CROSS-REFERENCE**

*CD_PLAY* plays specified tracks.

*CD_TRACK* allows you to find out which track is being played.

*CD_TRACKTIME* allows you to find out the total elapsed time on the current track.

*CD_RED2HSG* allows you to convert REDBOOK format to HSG Format.

*CD_HOUR*, *CD_MINUTE*, *CD_SECOND* allow you to convert REDBOOK format into a more understandable form.

## 10.14 CD_CLOSE

| | |
|---|---|
| Syntax | CD_CLOSE |
| Location | SMSQ/E for QPC |

This command closes the CD drive drawer, loading a CD if you have placed one in the drawer.

**CROSS-REFERENCE**

*CD_EJECT* opens the drawer.

*CD_PLAY* allows you to play a CD.

See *CD_INIT*.

## 10.15 CD_EJECT

| | |
|---|---|
| Syntax | CD_EJECT |
| Location | SMSQ/E for QPC |

This command opens the CD drive drawer and allows you to either place a new CD in the drive or to remove one.

You need to close the drawer before attempting to play the CD!

**CROSS-REFERENCE**

*CD_CLOSE* closes the CD drive drawer.

*CD_PLAY* allows you to play an Audio CD.

## 10.16 CD_FIRSTTRACK

| | |
|---|---|
| Syntax | CD_FIRSTTRACK |
| Location | SMSQ/E for QPC |

This function will return the track number of the first track on the CD currently in the player (this should always be 1).

**CROSS-REFERENCE**

*CD_LASTTRACK* allows you to find out the last track number.

## 10.17 CD_HOUR

| | |
|---|---|
| Syntax | CD_HOUR (address) |
| Location | SMSQ/E for QPC |

This function takes an address in REDBOOK format and tells you the number of hours (0..23) contained in that address.

**CROSS-REFERENCE**

*CD_MINUTE* and *CD_SECOND* allow you to find the number of minutes and seconds in a REDBOOK address respectively.

## 10.18 CD_HSG2RED

| | |
|---|---|
| Syntax | CD_HSG2RED (address) |
| Location | SMSQ/E for QPC |

There are two common formats used to address sectors on a CD directly. The standard format is REDBOOK format, which uses a time index to calculate the sector to address.

This time index is in the form $00MMSSFF where MM is the minute, SS the second and FF the frame.

There are 75 frames in one second.

The other format is HSG FORMAT where the sector is calculated by reference to the formula:

HSG=(minute*60+second)*75+frame

This function takes the address in HSG format and converts this to REDBOOK format.

**CROSS-REFERENCE**

*CD_RED2HSG* allows you to convert REDBOOK format addresses to HSG format.

*CD_HOUR*, *CD_MINUTE* and *CD_SECOND* allow you to find out the hours, minutes and seconds referred to by a REDBOOK address.

## 10.19 CD_INIT

| | |
|---|---|
| Syntax | CD_INIT [name$] |
| Location | SMSQ/E for QPC |

QPC is able to use a CD player linked to a PC in order to play Audio CDs at present.

You first of all need to initialise the CD drive by using this command. CD_INIT causes QPC to seach for a CD-ROM drive and initialise the driver.

You can either pass the name of the drive as a parameter or, if you do not use name$, then QPC will use the PC program MSCDEX (if present) to locate the CD-ROM Drive. MSCDEX can be loaded in the PC file AUTOEXEC.BAT if you wish, otherwise the CD drive name must appear in the PC file CONFIG.SYS.

**Example**

CD_INIT 'mscd001'

**NOTE 1**

This command will only be recognised once.

**NOTE 2**

The CD player commands and functions will not work if you have not loaded the PC's CD-ROM driver in config.sys, for example with the line:

DEVICE=C:\CD\CDROMDRV.SYS /D:MSCD001

**CROSS-REFERENCE**

*CD_PLAY* allows you to play CD Audio tracks.

*CD_EJECT* ejects a disk from the drive, or allows you to insert a new disk.

## 10.20 CD_ISCLOSED

| | |
|---|---|
| Syntax | CD_ISCLOSED |
| Location | SMSQ/E for QPC |

This function will return 1 (True) if the CD drawer is closed, otherwise it will return 0.

**Example**

```
100 IF NOT CD_ISPLAYING
110   IF NOT CD_ISCLOSED : CD_CLOSE
120   IF CD_ISINSERTED : CD_PLAY
130 END IF
```

**CROSS-REFERENCE**

*CD_CLOSE* closes the CD drawer.

## 10.21 CD_ISINSERTED

| | |
|---|---|
| Syntax | CD_ISINSERTED |
| Location | SMSQ/E for QPC |

This function will return 1 (True) if there is a CD in the CD-ROM drive and the drawer is closed, otherwise it will return 0.

**CROSS-REFERENCE**

See *CD_ISCLOSED*.

## 10.22 CD_ISPAUSED

| | |
|---|---|
| Syntax | CD_ISPAUSED |
| Location | SMSQ/E for QPC |

This function will return 1 (True) if the CD is paused (as opposed to stopped), otherwise it will return 0.

**CROSS-REFERENCE**

*CD_STOP* can be used to pause the CD.

*CD_RESUME* resumes playing a CD.

## 10.23 CD_ISPLAYING

| | |
|---|---|
| Syntax | CD_ISPLAYING |
| Location | SMSQ/E for QPC |

This function will return 1 (True) if an Audio CD is currently playing, otherwise it will return 0.

**CROSS-REFERENCE**

*CD_PLAY* allows you to play an Audio CD.

## 10.24 CD_LASTTRACK

| | |
|---|---|
| Syntax | CD_LASTTRACK |
| Location | SMSQ/E for QPC |

This function will return the track number of the last track on the CD currently in the player.

**CROSS-REFERENCE**

*CD_FIRSTTRACK* allows you to find out the first track number.

*CD_TRACK* tells you the track number currently playing.

## 10.25 CD_LENGTH

| | |
|---|---|
| Syntax | CD_LENGTH |
| Location | SMSQ/E for QPC |

This function will return the length of the Audio CD currently in the player in REDBOOK format.

**CROSS-REFERENCE**

*CD_LASTTRACK* allows you to find out the last track number.

*CD_HOUR*, *CD_MINUTE*, *CD_SECOND* convert REDBOOK format into a time.

## 10.26 CD_MINUTE

| | |
|---|---|
| Syntax | CD_MINUTE (address) |
| Location | SMSQ/E for QPC |

This function takes an address in REDBOOK format and tells you the number of minutes (0..59) contained in that address.

**CROSS-REFERENCE**

*CD_HOUR* and *CD_SECOND* allow you to find the number of hours and seconds in a REDBOOK address respectively.

## 10.27 CD_PLAY

| | |
|---|---|
| Syntax | CD_PLAY [start [,end]] |
| Location | SMSQ/E for QPC |

This command allows you to play the tracks on an audio CD once it has been initialised. If no parameters are specified, QPC will play the whole of the CD in the CD-ROM drive.

This command will not slow the operation of SMSQ/E and returns immediately that the CD starts playing.

The parameters allow you to specify the start and end tracks to be played. These parameters are given either as track numbers or as sectors in REDBOOK format (if bit 31 of the parameter is set). A sector on an Audio CD is 2352 bytes.

To set bit 31, add the value $80000000 or 2^31

**Examples**

CD_PLAY

plays the whole disk

CD_PLAY 10

play track 10 to the end of the disk

CD_PLAY 5,CD_TRACKSTART(5)+$80000000

play track 5 only.

A program which will play all of the tracks on an Audio CD in a random order:

```
100 INPUT 'Has the CD-ROM Drive already been initialised ? [y] ';an$
110 IF an$=='n': CD_INIT
120 IF NOT CD_ISINSERTED
130   IF CD_ISCLOSED : CD_EJECT
140   INPUT 'Place a CD in the drive and press <ENTER> ';an$
150   CD_CLOSE
160   IF NOT CD_ISINSERTED
170     PRINT 'NO CD LOADED ':PAUSE :STOP
180   END IF
190 END IF
200 tracks=CD_LASTTRACK-CD_FIRSTTRACK
210 DIM played% (tracks)
220 FOR i=1 to tracks
230   REPeat Floop
240     play=RND(1 TO tracks)
250     IF played%(play)=0: played%(play)=1: EXIT Floop
260   END REPeat Floop
270   CD_PLAY play,play
280   REPeat Ploop: IF NOT CD_ISPLAYING: EXIT Ploop
290 END FOR i
```

**CROSS-REFERENCE**

*CD_INIT* allows SMSQ/E to recognise a CD drive.

*CD_STOP* pauses playing

*CD_EJECT* opens the disk drawer to allow you to insert a new CD.

*CD_CLOSE* closes the disk drawer.

*CD_ISINSERTED* allows you to check if a CD is in the drive.

## 10.28 CD_RED2HSG

| | |
|---|---|
| Syntax | CD_RED2HSG (address) |
| Location | SMSQ/E for QPC |

This function converts a specified address in HSG format into REDBOOK format.

**CROSS-REFERENCE**

See *CD_HSG2RED* !

## 10.29 CD_RESUME

| | |
|---|---|
| Syntax | CD_RESUME |
| Location | SMSQ/E for QPC |

This command restarts the CD-ROM drive playing from the last track on which it was paused.

**NOTE**

If you had not previously paused the CD, then an error is reported.

**CROSS-REFERENCE**

*CD_STOP* allows you to pause a CD which is currently playing.

*CD_ISPAUSED* allows you to check if the CD has been paused.

## 10.30 CD_SECOND

| | |
|---|---|
| Syntax | CD_SECOND (address) |
| Location | SMSQ/E for QPC |

This function takes an address in REDBOOK format and tells you the number of seconds (0..59) contained in that address.

**CROSS-REFERENCE**

*CD_HOUR* and *CD_MINUTE* allow you to find the number of hours and minutes in a REDBOOK address respectively.

## 10.31 CD_STOP

| | |
|---|---|
| Syntax | CD_STOP |
| Location | SMSQ/E for QPC |

This command has one of two effects.

If an Audio CD is already playing, then the disk is paused.

If you have already paused the Audio CD, then a complete stop is performed.

**Example**

The following procedure brings the CD to a complete stop -

you cannot resume playing.

```
1000 DEFine PROCedure STOP_CD
1010 CD_STOP
1020 IF CD_ISPAUSED : CD_STOP
1030 END DEFine
```

**WARNING**

On some laptop PCs, it has been noted that if you are playing an Audio CD and close the case without issuing CD_STOP, when you re-open the case QPC will have crashed.

**CROSS-REFERENCE**

*CD_RESUME* allows you to resume playing an Audio CD that has been paused.

*CD_PLAY* allows you to play an Audio CD that is at a complete stop.

*CD_EJECT* opens the drive drawer.

*CD_CLOSE* closes the drive drawer.

## 10.32 CD_TRACK

| | |
|---|---|
| Syntax | CD_TRACK |
| Location | SMSQ/E for QPC |

This function returns the track number of which track on a CD is actually being played at present.

**CROSS-REFERENCE**

*CD_PLAY* plays specified tracks.

## 10.33 CD_TRACKLENGTH

| | |
|---|---|
| Syntax | CD_TRACKLENGTH (track) |
| Location | SMSQ/E for QPC |

This function returns the length of a specified track in HSG format.

**CROSS-REFERENCE**

*CD_TRACKTIME* allows you to find out the elapsed time on a track being played.

*CD_HSG2RED* converts the HSG format to REDBOOK format.

## 10.34 CD_TRACKSTART

| | |
|---|---|
| Syntax | CD_TRACKSTART (track) |
| Location | SMSQ/E for QPC |

This function returns the start address for a specified track in REDBOOK format.

**CROSS-REFERENCE**

*CD_TRACKLENGTH* allows you to find out the length of a track.

*CD_PLAY* allows you to play specified tracks

*CD_RED2HSG* converts the REDBOOK format to HSG format.

## 10.35 CD_TRACKTIME

| | |
|---|---|
| Syntax | CD_TRACKTIME |
| Location | SMSQ/E for QPC |

This function returns the actual elapsed time in REDBOOK format within the current CD track that is being played at present.

**CROSS-REFERENCE**

*CD_PLAY* plays specified tracks.

*CD_TRACK* allows you to find out which track is being played.

*CD_ALLTIME* allows you to find out the total elapsed time on the CD disk as a whole.

## 10.36 CEIL

| | |
|---|---|
| Syntax | CEIL (x) |
| Location | Math Package |

The function CEIL returns the closest integer to x which is greater than or equal to x (the 'ceiling' of x). Compare INT which returns the next integer which is less than or equal:

CEIL(12.75)=13 INT(12.75)=12 CEIL(-2.3)=-2 INT(-2.3)=-3

CEIL can handle numbers in the range -32768<x<=32768.

**Example**

A mechanic needs one and a half hours to replace the rusty exhaust of a car. If his rate of pay is £13 per hour, he will charge CEIL(13*1.5)=£20 for the job (excluding parts).

**NOTE**

The simplest way to get a true INTEGER function, where x is rounded up or down to the nearest integer is with INT(x+.5) which ensures that INT(12.75)=13 and INT(-2.3)=-2.

**CROSS-REFERENCE**

*INT*

## 10.37 CHANGE

| | |
|---|---|
| Syntax | CHANGE old_drv1$ TO new_drv2$ |
| Location | TinyToolkit |

This command allows you to rename directory devices. All directory device names are in the form xxxn_, where xxx identifies the drive type (eg. FLP) and n the drive number (1..8).

The most common drive types are:

- RAM - temporary internal ramdisk
- FLP - floppy disk drive (sometimes called FDK)
- MDV - microdrive
- MOS - permanent external ramdisk
- WIN - hard disk drive (sometimes HDK)

- NUL - null device, a dummy device
- DEV - universal devices (also PTH)

(Please see the Devices Appendix.)

CHANGE replaces the xxx part of a device name by a user defined name. This new name can already exist but both parameters must consist of three letters; the use of characters other than letters is possible but not recommended, eg:

CHANGE "flp" TO "<*>".

**Example**

CHANGE "ram" TO "mdv" makes the system believe that a ramdisk is a microdrive.

DIR mdv1_ will provide a directory of ramdisk 1, but the device ram1_ (or ram2_, etc.) is no longer recognised. The microdrives themselves cannot be accessed any more until you use: CHANGE "mdv" TO "ram" to restore the normal condition.

**NOTE**

If a device name is in ROM (eg. possibly mdv on QLs without floppy disk drives), the error -20 (read only) will be reported.

**CROSS-REFERENCE**

*FLP_USE* and *RAM_USE* work similarly.

## 10.38 CHANID

| | |
|---|---|
| Syntax | CHANID [(#ch)]Btool only or<br>CHANID (#ch)TinyToolkit |
| Location | BTool, TinyToolkit |

QDOS uses a different sort of channel number internally to those used by SuperBASIC. These so-called channel IDs have the advantage that two channels will never have the same channel ID, even if some channels have been closed for a long time.

The function CHANID expects an open SuperBASIC channel #ch (a default channel of #1 is allowed by Btool) and returns its current internal channel ID.

**Example**

```
100 OPEN#3,con_2x1
110 PRINT CHANID(#3)
120 CLOSE#3: OPEN#3,con_2x1
130 PRINT CHANID(#3)
140 CLOSE#3
```

**CROSS-REFERENCE**

*CHANID* is intended for use with *FILE_OPEN*.

*CHANNEL_ID* is the same as the Btool variant.

See SET_CHANNEL also.

## 10.39 CHANNELS

| | |
|---|---|
| Syntax | CHANNELS [#ch] |
| Location | BTool, Qsound, TinyToolkit |

The command CHANNELS list all channels which are currently open (including channels from any other job) to the given channel (default #1).

Each channel is listed with a channel number which can be used with CLOSE% and provides details of its size and position. Unfortunately, the name of the Job which owns the channel is not listed.

**NOTE**

The Tiny Toolkit and Qsound version of this command do not currently work with the Pointer Environment. The BTool version works to some extent.

**CROSS-REFERENCE**

*CLOSE%*, *JOBS* and *CHANID*

## 10.40 CHANNEL_ID

| | |
|---|---|
| Syntax | CHANNEL_ID [(#ch)] |
| Location | Turbo Toolkit |

This function is exactly the same as CHANID.

**CROSS-REFERENCE**

See*CHANID* and SET_CHANNEL.

## 10.41 CHAN_B%

## 10.42 CHAN_W%

## 10.43 CHAN_L%

| | |
|---|---|
| Syntax | CHAN_B% (#ch, offset) and<br>CHAN_W% (#ch, offset) and<br>CHAN_L |
| Location | CHANS (DIY Toolkit - Vol C) |

These three functions can be used to look at values within the channel definition block relating to the specified channel (#ch). You will need a good book on the QL's operating system to understand the various offsets, such as the QDOS/SMS Reference Manual (See section 18.7 to 18.9.3 in that book).

They allow you to read single bytes, words and longwords from the channel definition block (what is required depends upon the offset).

Extra offsets (negative numbers) are added by the Pointer Environment which can also be looked at by using these functions.

**Examples**

Instead of using SCR_BASE, you can use:

PRINT CHAN_L (#1,50)

to find the base address of the screen.

```
100 PRINT 'Window #1's size is';
110 PRINT CHAN_W% (#1,28);'x'; CHAN_W% (#1,30);'a'; CHAN_W% (#1,24);'x'; CHAN_W% (
↪#1,26)
```

**CROSS-REFERENCE**

*CHBASE* can be used to find out similar information.

## 10.44 CHARGE

| | |
|---|---|
| Syntax | CHARGE [task_file$] |
| Location | Turbo Toolkit |

This command starts up the Turbo Compiler and attempts to compile the program currently loaded in SuperBASIC Job 0.

It is similar to issuing the commands:

```
EXEC_W flp1_PARSER_TASK
EXEC flp1_CODEGEN_TASK
```

The default device which contains the Turbo compiler (PARSER_TASK and CODEGEN_TASK) can be configured with a special toolkit configuration program.

If you do not specify a task_file$, then the one which is configured is assumed to be the name of the new compiled file to be generated. This and several other defaults may be altered from the front panel which is generated by PARSER_TASK. The default settings on the front panel may also be configured and set using various directives such as TURBO_obfil.

The maximum length of the task_file$ is 12 characters. If a longer string is supplied, only the first 12 characters are used.

**Example**

CHARGE 'GENEALOGY'

**NOTE 1**

This command will not work on Minerva and SMS.

**NOTE 2**

The filename for the new task has never really worked correctly when passed as a parameter, if you specify a device as part of the filename. The filename becomes corrupted if this is the case.

**NOTE 3**

When you compile a program using TURBO, it is imperative that all of the machine code procedures and functions which are used by that program are linked into the machine. If you fail to do this, then an error will be reported when you try to run your compiled program using EXEC or EXEC_W for example.

This is different to QLiberator, which only checks whether each machine code function or procedure is linked in when (and if) it tries to use them whilst the compiled program is being run.

**CROSS-REFERENCE**

*DATA_AREA* and various TURBO_xxx directives exist, starting with *TURBO_diags* to allow you to specify various compilation options from within your program's source code.

Please also refer to *COMPILED*.

## 10.45 CHAR_DEF

| | |
|---|---|
| Syntax | CHAR_DEF font1,font2 |
| Location | SMSQ/E v2.57+ |

This command is very similar to the CHAR_USE command, except that instead of altering the fonts attached to a specified window, it sets the default fonts which are used for every new window channel that is opened after this command (unless they in turn define their own fonts).

The two parameters should point to an address in memory where a font in the QL font format is stored. If either parameter is 0, then that fount is reset to the standard system fount. If either parameter is -1, then CHAR_DEF will not affect that font.

Minerva users can achieve the same effect with the following:

```
110 Font0=PEEK_L (!124 !40)
120 Font1=PEEK_L (!124 !44)
130 POKE_L !124 !40, NewFont0, NewFont1
```

Note that you will need to store the addresses of the original QL ROM fonts (as in lines 110 and 120).

**NOTE 1**

The screen windows which are already open will not be affected.

**NOTE 2**

This command cannot affect a screen window which has been OPENed over the Network, unless issued on the Slave computer (on whose screen the window appears), before the window was OPENed over the Network.

**CROSS-REFERENCE**

*CHAR_USE*, *CHAR_INC*.

Please also refer to the Fonts Appendix.

## 10.46 CHAR_INC

| | |
|---|---|
| Syntax | CHAR_INC [#channel,] x_step,y_step |
| Location | Toolkit II, THOR XVI |

This command sets the horizontal (x_step) and vertical (y_step) distance between characters printed on a window (default #1). The standard values are the width and height of a character and are automatically

set by CSIZE.

CSIZE#2,0,0 performs an internal CHAR_INC#2,6,10.

Characters are generally based on a grid which measures 8x10 pixels, although the leftmost column was not available for fonts on pre-JS ROMs. Also, if you own a JSU ROM (an American QL), this grid size is reduced to 8x8, although programs would appear to run okay on the JSU ROM without modification (see MODE for further details).

**Example**

Would you like to print more characters to the screen than normal? You can either do this by defining smaller fonts or by writing characters on the screen closer together:

```
100 WINDOW 512,40,0,0:CLS
110 CSIZE 0,0: CHAR_INC 5,8:OVER 1
120 PRINT FILL$('.',102)
```

Window #1 now offers 5 rows and 102 columns instead of 4 rows and 85 columns, but text can only be read in overwrite mode (OVER 1). CHAR_INC 6,8 is the highest possible value which will allow text to be read without the need for OVER 1.

**WARNING**

Unless you have Minerva or Lightning installed (with _lngASLNG enabled), if you specify a character height less than the standard 10 pixels (for CSIZE x,0) for example, the strip printed will remain at ten pixels, and although the screen driver might detect that it is not necessary to scroll a window to fit the text on, it does not take account of the height of the strip, which could therefore fall out of the window (or into the system variables if your window is near the bottom of the screen).

**CROSS-REFERENCE**

*CSIZE*, *OVER*.

See also *TTINC*.

## 10.47 CHAR_USE

| | |
|---|---|
| Syntax | CHAR_USE [#ch,] font1,font2 |
| Location | Toolkit II, THOR XVI |

This command allows you to attach substitute fonts in QDOS format to the specified window channel (default #1).

CHAR_USE will attach the two fonts at addresses font1 and font2 to the window in place of the current system fonts.

When a character is printed, if it cannot be found at either font1 or font2, then the first character of the second font will be used.

To return to the current system fonts on the specified window, use font1=0 or font2=0 as appropriate.

If you use the value of -1 as one of the parameters, then that font attached to the specified channel will not be altered by this command.

**Example**

CHAR_USE #3,font_address,0

resets the first font in #3 to the font stored at font_address in memory.

**NOTE**

This command will have no effect on a window OPENed over the Network.

**CROSS-REFERENCE**

Please refer to the Fonts Appendix concerning QL fonts.

*CHAR_INC* allows you to alter the spacing between characters.

*CHAR_DEF* allows you to alter the default system fonts.

*S_FONT* performs the same function as *CHAR_USE*.

## 10.48 CHBASE

| | |
|---|---|
| Syntax | CHBASE [(#ch)] or<br>CHBASE (chidx%, chtag%) |
| Location | QBASE (DIY Toolkit Vol Q) |

CHBASE is a function which returns the start address of a window definition block. This block contains a wide range of information about a window, such as the size and colour settings. Refer to the QDOS Reference manual Section 18.7 and 18.9.1 (or similar) for further details.

The window can be either specified by its SuperBASIC channel number, eg: CHBASE(#2), where the default is #1, or the internal channel ID; which must be split into index (chidx%) and tag (chtag%) before being passed to CHBASE.

The latter syntax allows you to access the windows of jobs other than the current job.

Inside knowledge about the operating system is necessary to access these tables. Please refer to QDOS system documentation. The structure of the window definition block is different under Thors, original QLs and the Pointer Environment.

CHBASE returns small negative integers if an error occurs, representing the QDOS error code:

- -1 = Window is currently in use, eg. awaiting input.
- -6 = No such channel exists.
- -15 = It's a channel but not a window.

**Example 1**

The current INK colour is found at offset $46, so: INK 7: PRINT PEEK( CHBASE+ HEX('46') ) will print 7, because of the INK 7 command.

**Example 2**

It is usually not recommended to close and re-open SuperBASIC channel #0. The following lines check if this has happened, although they will only work under the SuperBASIC interpreter(!). You will find the condition in line 100 is always true for Minerva's MultiBASIC interpreters and SMS's SBASIC interpreters: this does no harm - the example is more or less just an example of the syntax of CHBASE...

```
100 IF CHBASE(0,0) <> CHBASE(#0) THEN
110   UNDER 1: PRINT "Warning": UNDER 0
120   PRINT "Channel #0 is not in it's original state."
130 END IF
```

**CROSS-REFERENCE**

*CBASE*.

See also *CHAN_B%* and related functions.

## 10.49 CHECK

| | |
|---|---|
| Syntax | oops = CHECK('name') |
| Location | DJToolkit 1.16 |

If name is a currently loaded machine code procedure or function, then the variable oops will be set to 1 otherwise it will be set to 0. This is a handy way to check that an extension command has been loaded before calling it. In a Turbo'd or Supercharged program, the *EXEC* will fail and a list of missing extensions will be displayed, a QLiberated program will only fail if the extension is actually called.

**EXAMPLE**

```
1000 DEFine FuNction CheckTK2
1010   REMark Is TK2 present?
1020   RETurn CHECK('WTV')
2030 END DEFine
```

## 10.50 CHECK%

| | |
|---|---|
| Syntax | CHECK% (integer$) |
| Location | CONTROL (DIY Toolkit Vol E) |

Coercion is the process of converting a string which holds a number into the actual number. It is a powerful in-built feature of SuperBASIC. This allows you to create input routines such as:

```
100 dage% = RND(10 TO 110)
110 INPUT "Your age [" & dage% & "]?" ! age$;
120 IF age$ = "" THEN
130   age% = dage%: PRINT age%
140 ELSE
150   age% = age$: PRINT
160 END IF
```

Although SuperBASIC coercion is very powerful, it does have its limits when non-numeric strings are entered. If age$ was "44", age%=age$ will assign 44 to age%. Even if the string was not really a number, eg. "44x5", SuperBASIC will simply ignore everything behind legal characters (ie. age%=age$ would assign 44 to age% still). However, if age$ contained something like "no thanks" it cannot be coerced and the program will fail with an 'error in expression' (-17).

The function CHECK% exploits the fact that SuperBASIC is obviously able to see the difference between a valid number or what comes close to that and nonsense. CHECK% carries out an explicit coercion for integer numbers: it will try to make a number from the supplied parameter in the same way as SuperBASIC would. However, CHECK% will not stop with an error for unusable strings, instead it returns -32768.

Although "-32768" is converted correctly to -32768, this value must be reserved because the program cannot know whether the input was illegal or -32768.

**Example**

Let's rewrite the above example for coercion with CHECK%. We have to replace the implicit coercion age%=age$ with age%=CHECK%(age$) and put INPUT into a loop:

```
100 dage% = RND(10 TO 110)
110 REPeat asking
120   INPUT "Your age [" & dage% & "]?" ! age$;
130   IF age$ = "" THEN
140     age% = dage%: PRINT age%
150   ELSE
160     age% = CHECK%(age$): PRINT
170     IF age% > -32768 THEN EXIT asking
180   END IF
190 END REPeat asking
```

**CROSS-REFERENCE**

*CHECKF* does the same as *CHECK%* but converts strings containing floating point numbers.

*WHEN ERRor* can trap the coercion failure.

See the Coercion Appendix also.

## 10.51 CHECKF

| | |
|---|---|
| Syntax | CHECKF (float$) |
| Location | CONTROL (DIY Toolkit Vol E) |

Just like CHECK%, the function CHECKF takes the specified string and coerces it to a number. This time, however, the number returned will be a floating point rather than an integer as returned by CHECK%.

CHECKF works just like CHECK% except that a return value of -1E600 signifies unacceptable strings.

**CROSS-REFERENCE**

*CHECK%* and *TTEFP* are worth a look.

## 10.52 CHK_HEAP

| | |
|---|---|
| Syntax | CHK_HEAP |
| Location | SMSQ/E |

This command is used to check whether the heap has become corrupted - we have no real details over its working as it is undocumented.

## 10.53 CHR$

| | |
|---|---|
| Syntax | CHR$ (code) |
| Location | QL ROM |

This function returns the character associated with the given code.

The QL ROM character set is actually only in the range 0…255, although code can be anything in the range -32768…32767. The least significant byte of the supplied parameter is used, ie. code && 255.

**Examples**

PRINT CHR$(100) and PRINT CHR$(1636)

both return 'd'.

A short function to convert any lower case letters in a given string to upper case:

```
100 DEFine FuNction UP$(a$)
110 LOCal U$
115 U$=a$
117 IF a$='':RETurn ''
120 FOR i=1 TO LEN(a$)
130   IF CODE( a$(i) )>96:IF CODE( a$(i) )<123:U$(i)=CHR$( CODE( a$(i) )-32 )
140 END FOR i
150 RETurn U$
160 END DEFine UP$
```

**NOTE**

The THOR XVI limits code to the range 0…255.

**CROSS-REFERENCE**

See *CODE* and also please refer to the Characters section of the Appendix.

## 10.54 CIRCLE

| | |
|---|---|
| Syntax | CIRCLE [#ch,] x,y,radius [,ratio,ecc] $^{*}$[;$x^{i}$,$y^{i}$,$radius^{i}$ [,$ratio^{i}$,$ecc^{i}$]]$^{*}$ |
| Location | QL ROM |

This command allows you to draw a circle of the given radius with its centre point at the point (x,y).

The positioning and size of the circle will actually depend upon the scale and shape of the specified window (default #1).

The co-ordinates are calculated by reference to the graphics origin, and the graphics pointer will be set to the centre point of the last circle to be drawn on completion of the command.

If any parts of the circle lie outside of the specified window, they will not be drawn (there will not be an Overflow Error).

If the parameters ratio and ecc are specified, this command has exactly the same effect as ELLIPSE.

This command will actually allow you to draw multiple circles by including more sets of parameters. Each additional set must be preceded by a semicolon (unless the preceding circle uses five parameters). This means that these commands are all the same:

```
CIRCLE 100,100,20,1,0,50,50,20
CIRCLE 100,100,20;50,50,20
CIRCLE 100,100,20:CIRCLE 50,50,20
```

Although the FILL command will allow you to draw filled circles on screen (in the current ink colour), you will need to include a FILL 1 statement prior to each circle if they are to appear independently on screen (this cannot be achieved when using this command to draw multiple circles!). If this rule is not followed, then any points which lie on the same horizontal line (even though they may be in different circles) will be joined.

**Example**

Try the following for an interesting effect:

```
100 WINDOW 448,200,32,16: MODE 8
110 PAPER 0: CLS
120 SCALE 200,-100,-100
130 INK 4:CIRCLE -50,-50,5
140 FOR i=1 TO 350
150   INK RND(7): FILL 1
160   CIRCLE_R 5-(i MOD 10),15-(i MOD 30),20
170 END FOR i
```

**CROSS-REFERENCE**

Please refer to *ELLIPSE* for further information on the ratio and ecc details.

## 10.55 CIRCLE_R

| | |
|---|---|
| Syntax | CIRCLE_R [#ch,] x,y,radius [,ratio,ecc] $^{*}$[;$x^{i}$,$y^{i}$,$radius^{i}$ [,$ratio^{i}$,$ecc^{i}$]]$^{*}$ |
| Location | QL ROM |

This command draws a circle relative to the current graphics cursor. See CIRCLE.

**CROSS-REFERENCE**

Please refer to *ARC_R*. *ELLIPSE_R* is exactly the same as this command.

## 10.56 CKEYOFF

| | |
|---|---|
| Syntax | CKEYOFF |
| Location | Pointer Interface (v1.23 or later) |

Normally, the Pointer Interface will recognise the cursor keys in the same way as it recognises the mouse, thus allowing you to move the pointer around the screen using the keyboard.

You may however prefer that the cursor keys had no effect on the pointer - the solution is simple - just use the command CKEYOFF.

**NOTE**

There were problems with this command prior to v1.56.

**Note 2**

At some point CKEYOFF required a channel number parameter, ie CKEYOFF #channel. If you don't specify a channel number the command falls back on the current S*BASIC's channel #0.

Unfortunately, if there isn't already a channel #0 open, for example, a program might have opened its first channel using:

1500 main = FOPEN("con_")

In this case, the the CKEYOFF command will open a new console channel and then try to set the window size of that channel to the default size of 256x62. If, however, the program had already OUTLN'ed #main to a smaller size, in any dimension, than 256x62, then this call will fail.

And here's the bug: Instead of closing the failed console channel, the command simply returns without error. This leaves a "dangling" console channel open without an S*BASIC handle. And each repeated call to CKEYOFF/CKEYON opens another such channel!

As of ptr_gen 2.07 and SMSQ/E 3.39, this bug should be fixed, in that an "Out of Range" error will be returned should the situation described above arise.

The workaround for earlier versions is: Always use this command with a channel number *unless* there is definitely an available console channel #0!

The bug fix is that, if the circumstances described pertain, the call will return with an "Out of Range" error.

**CROSS-REFERENCE**

*CKEYON* tells the Pointer Interface to recognise the cursor keys again.

## 10.57 CKEYON

| | |
|---|---|
| Syntax | CKEYON [#channel] |
| Location | Pointer Interface (v1.23 or later) |

See *CKEYOFF*.

**NOTE**

There were problems with this command prior to v1.56.

**Note 2**

At some point CKEYON started to have to take a channel number, ie CKEYON #channel.

If you dont give a channel number the commands try to open their own channel - that hybrid number 0/1, as do a number of IO commands (at least in SMSQ/E). All well and good. But if you, say, open your first channel as number 3, as happens automatically if your first channel in a daughter SBASIC or Qlib compiled job, and you open it with ch = FOPEN("con_") then, if that channel is OUTLiNed to be smaller than the default channel, the call fails and returns to the routine - which gives up. But doesnt say

anything! However, now there is dangling, open channel in the main channel table that SBASIC doesnt know about. So next time you use this command yet another channel is opened in the main channel table! And so it goes on ad finitum until the job is killed - or the system chokes.

This is a rare circumstance, admittedly, but it is still a bug.

The bug fix is that if the circumstances described pertain, the call will return with an Out of Range error.

## 10.58 CLCHP

| | |
|---|---|
| Syntax | CLCHP |
| Location | Toolkit II, THOR XVI, Btool |

A BASIC program can reserve space in the common heap with ALCHP. The command CLCHP removes all space which has been grabbed using ALCHP and returns it to the common heap so that it can be used for other purposes.

**CROSS-REFERENCE**

*ALCHP* reserves areas of the common heap, and *RECHP* releases a specified part of the common heap.

Compare *RESERVE* and the Btool variant of *ALCHP*.

*NEW* and *LOAD* also release areas of the common heap.

## 10.59 CLEAR

| | |
|---|---|
| Syntax | CLEAR |
| Location | QL ROM |

This command forces all variables to be cleared meaning that the computer will no longer remember their values.

This does not affect SuperBASIC functions or resident keywords, for example, PRINT PI will always return 3.141593.

On non-SMS machines, if a variable is PRINTed, which has not yet been assigned a value, an asterisk appears on screen. If you try to *use* a variable which has not yet been assigned a value, then an error will occur (normally error in expression (-17)).

If Toolkit II is present (or you are using Minerva or a THOR XVI), any valid WHEN structures are also suspended by the CLEAR command.

Adding CLEAR before a program is run ensures that all variables used in a program will be defined properly. While developing a large program in BASIC it may sometimes be helpful to set an essential variable directly in the command line and not as a static statement in the listing.

**Example**

The following lines will produce a different output depending on whether they have been run before or not:

```
100 PRINT a
110 a=5
120 PRINT a
```

The first run shows… * 5 This is because the contents of a were not defined until line 110 was reached.

The second time, a was still set and so the output is slightly different… 5 5

**NOTE**

CLEAR may cause some problems on pre Minerva ROMs if it is issued after having deleted a PROCedure or a FuNction in a SuperBASIC program which appeared as the last thing in a program. This is fixed by Toolkit II.

**SMS NOTE**

Variables which have not been assigned a value on SMS will return 0 (zero) if a numeric variable or otherwise an empty string - an error will therefore not occur if you try to use such a variable.

On a machine fitted with SMS the example would therefore have printed 0 5 on the first run, and 5 5 on the second.

**CROSS-REFERENCE**

*CLOSE*, *CLEAR_HOT*, *CLCHP*, *CLRMDV*, *RUN*.

## 10.60 CLEAR_HOT

| | |
|---|---|
| Syntax | CLEAR_HOT key |
| Location | TinyToolkit |

This command deletes a hotkey defined with the HOT command and releases the memory used to set up the hotkey back to QDOS' memory management.

**NOTE**

CLEAR_HOT works okay, but in most cases the memory released by this command is not recognised by the system as being free memory and therefore cannot be re-used without resetting the system.

**CROSS-REFERENCE**

See *HOT* on how to define a hotkey.

Use *FREE*, *FREE_MEM* to check the actual available memory.

## 10.61 CLIP%

| | |
|---|---|
| Syntax | CLIP% (#channel) |
| Location | CLIP (DIY Toolkit - Vol S) |

This function can be used to read characters from the QL's screen.

In order for the function to work, you will need to OPEN a window over that part of the QL's screen which you wish to read and ensure that it is in the correct MODE and has UNDER, CSIZE and CHAR_INC

set to the same values as were used to create that part of the screen. You will also need to ensure that the same font is being used by the window which you have OPENed. The window should be defined so that any text written to that window would precisely match the text on screen (except for colour).

Due to the way in which QL's work, this means that CLIP% can be used to read user-defined characters from the screen, for example, where in games some of the font has been redefined to represent symbols in the game.

The function will then try to read a character from the current cursor position and return its character CODE. It can be used to read any character in the range 0...255 (except CHR$(10) which does not appear on screen).

The DIY Toolkit includes an example of a program which uses this function to create a clip board for reading text from a program running on the QL. It uses CHAN_W% and similar functions to read the existing settings of the window of a target program.

However, this function is really of most use when used within your own programs, possibly to detect collisions in a game between objects.

**Example**

The following short routine could be used to read the name of a disk in flp1_ (provided that the directory was not longer than one page):

```
10 DIR flp1_
20 FOR i=0 TO 20
25 AT #1,1,i
30 PRINT #2,CLIP$(#1);
40 END FOR i
```

**NOTE 1**

Although this works on all QL implementations, the code will not currently work with resolutions bigger than 512x256 pixels.

**NOTE 2**

If you want to read characters from a window of a program whilst the THOR XVI's windowing environment, or the Pointer Environment is running, you will have to switch off the windowing environment before the program in question is loaded, using POKE SYS_VARS+133,1 on the THOR or EXEP flp1_program,u under the Pointer Environment.

**NOTE 3**

The main problem with these functions is that some programs do not use standard fonts (or attach fonts to a window using non-standard techniques). Some additional fonts are supplied with DIY Toolkit which may help in this respect.

**CROSS-REFERENCE**

See the Fonts Appendix about changing QL fonts.

*CHAR_USE* and *S_FONT* allows you to set the font used by a window.

See also *CLIP$*.

## 10.62 CLIP$

| | |
|---|---|
| Syntax | CLIP$ (#channel) |
| Location | CLIP (DIY Toolkit - Vol S) |

This function is very similar to CLIP% except that it returns the actual character which appears on screen rather than the character code.

**NOTE**

The same notes apply to this function as to CLIP%.

**CROSS-REFERENCE**

See *CLIP%*.

## 10.63 CLOCK

| | |
|---|---|
| Syntax | CLOCK [#channel] [,format$] |
| Location | Toolkit II, THOR XVI |

The command CLOCK creates a multitasking digital clock job named Clock. If no channel parameter is stated, CLOCK will open its own window (con_60x20a448x206), which is intended for F1-monitor mode (see WMON), otherwise the given channel will be used.

Format$ is optional and is used to define how the clock will appear on screen. It can contain any text you desire (except for the characters % or $), but there are certain special characters (see below) which allow you to alter the way in which the clock is presented; so CLOCK "TEA AT 4" might remind you when tea time is, but will have no effect on the display of the clock.

The format is defined by using certain set series of strings. The following special characters will affect the way in which the clock is displayed (the default format string is "$d %d $m %h:%m:%s" which is ideal) :

- %d Day of month - 2 digits
- $d Day of week - 3 characters
- %h Hour (24h) - 2 digits
- $m Month - 3 characters
- %m Minute - 2 digits
- %s Seconds - 2 digits
- %y Year - 2 digits (last two digits)
- %c Century - 2 digits (see note 4 below)

A newline can be inserted by either padding out the string with spaces or by adding CHR$(10) inside the string.

**Example**

CLOCK #2,'Date: %d $m %y' & chr$(10) & 'Time: $d %h:%m'

**NOTE 1**

There is no difference between upper case and lower case letters, so %d has the same effect as %D. However, do watch the difference between $m and %m!

**NOTE 2**

Any attempt to open a clock in channel #0 will be ignored and the default window opened.

**NOTE 3**

Unfortunately for Pointer Environment users, there is no way of 'unlocking' the clock so that it can operate alongside other Jobs. On the THOR XVI this is alleviated by ensuring that the Job is always owned by Job 0.

**NOTE 4**

v2.25+ of Toolkit II introduced a further special character for use in the format string. This is %c, which returns the first two digits of the year, for example %c%y will print the current year as four digits.

**NOTE 5**

On v6.41 of the THOR XVI, if CLOCK has to open its own window, this window is in fact owned by SuperBASIC rather than the CLOCK task. This means that if CLOCK is removed other than by using NO_CLOCK, (eg. with RJOB) the channel can be left open.

**CROSS-REFERENCE**

Use *SDATE* or *ADATE* to set the system date and time.

*DATE$* and *DATE* return the current time.

*NO_CLOCK* removes the *CLOCK* on the THOR.

## 10.64 CLOSE

| | |
|---|---|
| Syntax | CLOSE #channel or<br>CLOSE #channel1 [, #channel2 . . . ] (Toolkit II, Btool & Minerva v1.81+) or<br>CLOSE (Toolkit II, THOR & Minerva v1.81+, BTool) |
| Location | QL ROM, Toolkit II, BTool,. THOR |

CLOSE is a procedure which closes a specified channel, (or even several channels if the second or third variant is used). The contents of that channel will however remain unchanged.

The second variant allows any number of specified channels to be closed at the same time and the third closes all channels with channel numbers of #3 or above.

Every CLOSE command will first flush the contents of internal buffers to ensure that all information has been passed to the channel before it is closed.

**Examples**

```
CLOSE#3
CLOSE#n
CLOSE #1
```

(continues on next page)

(continued from previous page)

```
CLOSE#8,#3,#6
CLOSE
```

**NOTE 1**

On Minerva pre v1.81 and other ROMs, unless Toolkit II is installed, CLOSE will report 'channel not open' if the channel is not open. Toolkit II and later versions of Minerva stop this from happening.

**NOTE 2**

There is a harmless bug in Toolkit II's CLOSE. This will report error -15 (bad parameter) if channel #32767 was opened and CLOSE issued without parameters, or even if you use the explicit command CLOSE #32767 (unless you have SMS). Although #32767 will still be closed successfully, any further attempt to use CLOSE without parameters will continue to report error -15 until the program is cleared out with NEW, LOAD or LRUN.

**NOTE 3**

On Minerva, if you have Lightning installed, then unless you CLOSE channels in the opposite order to that in which they were OPENed, you may end up with several CLOSEd windows which are still visible on screen. This will only disappear when another channel with the same channel number is opened. The Pointer Interface and SMS cure this.

**NOTE 4**

Unless you have a THOR XVI or Minerva (without SMS), do not CLOSE a network out (eg. NETO_1) channel unless you have written something to it. The machine will lock up if you do so be warned! On a THOR, the system will lock up for 30 seconds and then report an 'Xmit Error'. On Minerva, you will need to press <CTRL><SPACE>.

**NOTE 5**

QL ROMs (pre MG) had problems in closing ser2 - they tended to close one serial channel for output and the other for input instead!

NOTE 6:

**NOTE 6**

If you are writing to a file (especially on a microdrive cartridge), ensure that the drive has finished turning after issuing the CLOSE command, before trying to access the file (otherwise you may find that all of the changes are not present!). The other solution is to FLUSH the file before CLOSEing it.

**MINERVA NOTE**

CLOSE #1 will also remove a MultiBasic job in certain instances - see appendix on Multiple Basics.

**WARNING**

Although under the interpreter, channel #0 (the command window) and channels #1 and #2 can be closed, this will lock up the SuperBASIC interpreter. It does no harm at all in compiled programs.

Minerva and SMS prevents this from being disastrous, but some programs may behave a little strangely on the newly opened #0. If you use CLOSE #0 from within a MultiBASIC or one of SMS's SBASICs, this will remove the MultiBASIC (or SBASIC) Job.

**CROSS-REFERENCE**

*OPEN*, *CHANNELS*, *CLOSE% SCR_STORE* and related commands can be used to provide the QL with a windowing environment whereby the contents of the screen are restored when a window is *CLOSE*d.

## 10.65 CLOSE%

| | |
|---|---|
| Syntax | CLOSE% n |
| Location | BTool, TinyToolkit |

The command CLOSE% allows you to close a channel which is specified using the channel number listed when you use the CHANNELS command. This thus allows you to close channels owned by other Jobs.

**WARNING**

If you close the channel of a job, this can lock up that job. Ensure that you know the consequences of your actions!

**CROSS-REFERENCE**

*CHANNELS*, *CLOSE*

## 10.66 CLRMDV

| | |
|---|---|
| Syntax | CLRMDV n |
| Location | TinyToolkit, Btool |

This command forces the QL to forget that it had already read a cartridge in the given microdrive mdvn_. This could be necessary if cartridges are exchanged between QLs, otherwise one of the QLs may not find a file written by another QL on a cartridge. Such problems do not exist with floppies or any other media.

**Example**

CLRMDV 2

**CROSS-REFERENCE**

For *RAND*, *CLRMDV* is very useful.

See also *DEL_DEFB* which performs a similar task.

## 10.67 CLS

| | |
|---|---|
| Syntax | CLS [#chan,] [cls_type] |
| Location | QL ROM |

This command is normally used to clear all or part of the specified window (default #1) to the current paper colour for that channel (this is not affected by OVER). CLS does not affect a border attached to a window.

The cls_type can be used to specify which area of the window is to be cleared (the default is 0). This can have the following standard values, which have different effects depending upon the current position of the text cursor:

- 0 Clear the whole window
- 1 Clear the window above the cursor line

- 2 Clear the window below the cursor line
- 3 Clear the whole cursor line
- 4 Clear the window from the cursor position to the right-hand end of the cursor line

After using this command, the text cursor is placed at the top left-hand corner of the window (if cls_type=0) or at the current cursor position for other values.

Except under SMS and on the THOR XVI, most systems allow you to use other values for cls_type to access various TRAP #3 system utilities. The way in which the appropriate value of cls_type is calculated is by taking the value of D0 which would be used in machine code and subtracting 32 from this. If this gives a negative result, then add this negative result to 128.

For example, to move the cursor back one space, in machine code you would use the call IOW.PCOL (D0=19). 19-32=-13, therefore:

CLS #3,128-13 moves the cursor back one space in #3.

You must however be aware of using CLS 98 (IO.FLINE) on pre JS ROMs, since this tended to leave the cursor switched on!

**NOTE 1**

On pre MG ROMs CLS is likely to fail if the window is smaller than the cursor.

**NOTE 2**

The THOR XVI only allows cls_type to be in the range 0..4.

Under SMS, if cls_type is more than 4, then CLS uses cls_type MOD 4.

**NOTE 3**

Some of the additional values of cls_type can actually cause the computer to crash, whilst others will merely report an error.

**CROSS-REFERENCE**

*AT* and *PRINT* position the text cursor.

*PAPER* alters the current paper colour.

*SCROLL* and *PAN* also allow you to access various system utilities.

A much safer way to access system utilities is to use *IO_TRAP*, *MTRAP*, *QTRAP* and *BTRAP*.

For details of the various TRAP #3 system utilities refer to the QDOS/SMS Reference Manual (Section 15) or similar.

## 10.68 CLS_A

| | |
|---|---|
| Syntax | CLS_A |
| Location | BeuleTools |

This command forces all windows currently OPENed by SuperBASIC or belonging to the current job to be cleared and given a border (width 1, colour 255). This works on channels opened on Minerva's dual screens.

**CROSS-REFERENCE**

*CLS* clears a single window without changing window attributes, the border in this case.

## 10.69 CMD$

| | |
|---|---|
| Syntax | CMD$ |
| Location | SMS, Minerva |

This function can be used from within SMS SBASICs, Minerva MultiBASICs and compiled programs (not SuperBASIC Job 0) to read a string passed to the program when it was initiated, with the command EX (or similar).

The string appears after the EX command, preceded with a semicolon.

**Example**

Create a program to load in Xchange and set its default drives and memory, something akin to:

```
10 xch_data$=DATAD$:xch$=PROGD$
20 data_space=300
30 x$=CMD$
40 IF x$<>''
45   datpos='\' INSTR x$
50   IF datpos:data_space=x$(datpos+1 TO)
55   IF datpos>5:x$=x$(1 TO datpos-1)
74   dr1=',' INSTR x$
75   IF dr1<6
80     IF dr1=0:PROG_USE x$(1 TO):ELSE IF dr1<LEN(x$-4):DATA_USE x$(dr1+1 TO)
90   ELSE
100     PROG_USE x$(1 TO dr1-1)
110     IF dr1<LEN(x$)-4:DATA_USE x$(dr1+1 TO)
120   END IF
140 END IF
150 EX xchange;data_space
160 DATA_USE xch_data$
170 PROG_USE xch$
```

Save this as FLP1_XCHANGE_BAS (or similar).

Now, to pass the relevant parameters all you need do under SMS is enter the line:

EXEC flp1_XCHANGE_BAS;'win1_XCHANGE_,flp2_\200'

Minerva treats the string slightly differently - see EX for an explanation of the following command which achieves the same:

EXEC pipep;'flp1_XCHANGE_BAS>win1_XCHANGE,flp2_\200'

This will execute win1_XCHANGE_xchange with the help file to be loaded from win1_XCHANGE_ , the data files being loaded from flp2_ and a dataspace of 200K.

**NOTE 1**

In SMS pre v2.60, you could not directly slice CMD$ - copy it to another string variable first, as in the example.

**NOTE 2**

You cannot use this command in TURBO compiled jobs - use OPTION_CMD$ instead.

**CROSS-REFERENCE**

See *EXEC* and *EXEC_W*.

## 10.70 CODE

| | |
|---|---|
| Syntax | CODE (character$) |
| Location | QL ROM |

This function returns the internal code used to represent the given character$ (this will be a value between 0 and 255).

If the supplied parameter is more than one character in length, the code of the first character will be returned. The result 255 represents the ALT key, although this will only be produced with the statement PRINT CODE(INKEY$) if the <ALT> key is being pressed together with a second key, in which case the code of the second key quickly follows. If character$ is a nul string, CODE will return 0.

**Example 1**

PRINT CHR$(CODE('Alpha'))

will print 'A'.

**Example 2**

A short program to reveal the code of the current key being pressed (with special code to trap the instance of the ALT key being pressed):

```
100 REPeat loop
110   AT 0,0: a$ = INKEY$(#1, -1)
120   IF CODE(a$) = 255
130     PRINT 'ALT+' & CODE(INKEY$) & '  '
140   ELSE PRINT CODE(a$);'      '
150   END IF
160 END REPeat loop
```

Try replacing lines 110 to 150 with: 110 AT 0,0: PRINT CODE(INKEY$(-1))

**CROSS-REFERENCE**

Please refer to the Characters section of the Appendix for a full list of the characters and their internal codes.

## 10.71 CODEVEC

| | |
|---|---|
| Syntax | CODEVEC (name$) |
| Location | ALIAS (DIY Toolkit - Vol A) |

This function is very similar to KEY_ADD in that it returns the address in memory where the machine code for a machine code Procedure or Function is stored (useful for debugging programs with Qmon or similar machine code monitor).

If the Machine Code Procedure or Function with the given name$ does not exist, then a 'Not Found' error is reported.

**CROSS-REFERENCE**

See *KEY_ADD* and *ELIS*.

## 10.72 COL

| | |
|---|---|
| Syntax | COL(x, y) |
| Location | HCO |

COL is a function which returns the colour of a given screen pixel (specified in absolute co-ordinates). The colour is however not coded in the usual way - the return value of COL is either 0, 1, 2 or 3 (representing the four true colours which can displayed in MODE 4, ie. black, red, green and white).

**Example**

```
100 WMON: LIST#2
110 xmin% = 0: xmax% = 100
120 ymin% = 0: ymax% = 100
130 FOR x% = xmin% TO xmax%
140   FOR y% = ymin% TO ymax%
150     c% = 2 * COL(x%,y%) + 1
160     BLOCK 1, 1, x%-xmin%, y%-ymin%, c%
170   END FOR y%
180 END FOR x%
```

Unless you are using Minerva or SMS, replace x% and y% by x and y.

**NOTE**

COL will return meaningless data unless the screen is located at address 131072, is in MODE 4, and uses a 512 x 256 resolution.

**CROSS-REFERENCE**

*SET* draws a screen pixel.

## 10.73 COLOUR_NATIVE

| | |
|---|---|
| Syntax | COLOUR_NATIVE [#ch] |
| Location | SMSQ/E v2.98+ |

COLOUR_NATIVE is a command used to select the colour palette to be used from within the Extended Colour Drivers provided with SMSQ/E v2.98+ on the Q40/Q60, QXL, QPC and Aurora.

A valid window channel must be open, default #1 (or #0 on a SBASIC with only #0 open), although one may also be supplied as #ch.

This command is similar to COLOUR_PAL, but allows you to use 256 colours on Aurora, or 65536 colours on QXL, QPC and the Q40/Q60, by selecting the native colour mode of the hardware.

Colour parameters supplied to commands such as INK are defined in native colours and therefore their effect will depend upon the hardware itself (Appendix 16 contains details of the first 256 colours and their Native Colour Values in decimal, hexadecimal and binary for use with the INK command or similar).

**NOTE**

MODE commands have no effect under the Extended Colour Drivers.

**CROSS-REFERENCE**

Refer to *COLOUR_PAL* for more details.

## 10.74 COLOUR_PAL

| | |
|---|---|
| Syntax | COLOUR_PAL [#ch] |
| Location | SMSQ/E v2.98+ |

COLOUR_PAL is a command used to select the colour palette to be used from within the Extended Colour Drivers provided with SMSQ/E v2.98+ on the Q40/Q60, QXL, QPC and Aurora.

This command requires the Extended Colour Drivers to have been loaded when SMSQ/E started (set by configuration or chosen from the start-up menu on QPC). It will not have any effect upon programs already loaded into the system.

A valid window channel must be open, default #1 (or #0 on a SBASIC with only #0 open), although one may also be supplied as #ch.

COLOUR_PAL selects the PAL colour mode, allowing 256 colours to be used. After using this command, the effect of the colour parameters supplied to commands such as INK will depend upon the table which appears in Appendix 16 - use the PAL colour value given for each colour (this is hardware independent).

As a result, code such as that given in the example below is required to check on the colour scheme currently in use and adapt the program accordingly.

**Example**

```
100 REMark Make sure the program is in the right mode for Standard QL/
↪Extended Colours
110 col_sys%=0:h$=VER$
120 IF RMODE=8:MODE 4
130 IF RMODE=16:col_sys%=1:REMark Aurora - Extended Colour Drivers
140 IF RMODE=32:col_sys%=3:REMark QXL/QPC - Extended Colour Drivers
150 IF RMODE=33:col_sys%=2:REMark Q40 - Extended Colour Drivers
160 :
170 REMark Select Appropriate colour scheme
180 IF h$='HBA':IF col_sys%<>0:COLOUR_PAL
190 SELect ON col_sys%
200   =0:BLACK=0:WHITE=7:RED=2:GREEN=4:       REMark Four colours available
210   =REMAINDER :BLACK=0:WHITE=1:RED=2:GREEN=3:REMark 256 colours available
```

(continues on next page)

(continued from previous page)

```
220 END SELect
230 PAPER BLACK:INK GREEN
```

**NOTE 1**

The 256 colours produced under COLOUR_PAL on non-Aurora machines may be changed to allow any 24-bit colour using the command PALETTE_8. This will not work on Aurora, which has display hardware limited to 256 colours.

**NOTE 2**

MODE commands have no effect under the Extended Colour Drivers. RMODE will always report 16 on Aurora, 32 on QXL/QPC and 33 on the Q40/Q60 if the Extended Colour Drivers are in use.

**CROSS-REFERENCE**

Refer to Appendix 16 and *INK* for more details.

*COLOUR_QL*, *COLOUR_NATIVE* and *COLOUR_24* are all similar.

*PALETTE_QL* and *PALETTE_8* affect colour palettes.

*BGCOLOUR_QL* and *BGCOLOUR_24* can be used to alter the desktop colour of the main screen.

DISP_COLOUR can be used to switch between Extended Colour Drivers and the Standard Colour Drivers.

## 10.75 COLOUR_QL

| | |
|---|---|
| Syntax | COLOUR_QL [#ch] |
| Location | SMSQ/E v2.98+ |

COLOUR_QL is a command used to select the colour palette to be used from within the Extended Colour Drivers provided with SMSQ/E v2.98+ on the Q40/Q60, QXL, QPC and Aurora.

A valid window channel must be open, default #1 (or #0 on a SBASIC with only #0 open), although one may also be supplied as #ch.

This command is similar to COLOUR_PAL, but selects an 8 colour mode, with colours from 0…7 as per the original QL MODE 8 (although all 8 colours remain available for programs which presume MODE 4).

This can cause some slight incompatibility problems, due to programs which presume that under MODE 4, INK 3 would produce Red (for example) - under COLOUR_QL it will now produce Magenta.

**NOTE 1**

The eight colours produced under COLOUR_QL may be changed to allow any colour supported by the hardware using the command PALETTE_QL.

**NOTE 2**

MODE commands have no effect under the Extended Colour Drivers.

**CROSS-REFERENCE**

Refer to *COLOUR_PAL* for more details.

*PALETTE_QL* includes a way of overcoming the incompatibility problems with old *MODE*.. 4 programs.

## 10.76 COLOUR_24

| | |
|---|---|
| Syntax | COLOUR_24 [#ch] |
| Location | SMSQ/E v2.98+ |

COLOUR_24 is a command used to select the colour palette to be used from within the Extended Colour Drivers provided with SMSQ/E v2.98+ on the QXL and QPC, providing a good graphics card is installed.

A valid window channel must be open, default #1 (or #0 on a SBASIC with only #0 open), although one may also be supplied as #ch.

This command is similar to COLOUR_PAL, but allows you to specify colours directly using the 24 bit colour mode, thus allowing 16777216 (2^24) colours on screen at the same time.

Although the command does work on hardware which does not support a 24 bit graphics mode, the specified colours have to be adapted to fit into the memory available for each pixel (eg 8 or 16 bits). This can cause inaccuracies and unpredictable results - COLOUR_NATIVE is preferable in such circumstances.

**CROSS-REFERENCE**

Refer to *COLOUR_PAL* and *COLOUR_NATIVE* for more details.

*PALETTE_QL*, *PALETTE_8* and *BGCOLOUR_24* all use the 24 bit table to describe colours.

## 10.77 COMMAND_LINE

| | |
|---|---|
| Syntax | COMMAND_LINE |
| Location | Turbo Toolkit |

This command is really only of any use with the TYPE_IN command. It selects the SuperBASIC command line (#0) so that anything passed with TYPE_IN is automatically entered into that channel (as if it were typed).

Note that COMMAND_LINE cannot have any effect if SuperBASIC is doing something or if the job which uses the command was started with EXEC_W or similar.

**NOTE 1**

COMMAND_LINE pre v3c27 does not seem to work correctly on all versions of the QL ROM.

**NOTE 2**

Two files called TurboFix_bin and MiniCommdLin_bin can be used to allow COMMAND_LINE to select the command line of a Minerva MultiBASIC - this relies on the MultiBASIC being the job which uses the COMMAND_LINE command. Some early versions of TurboFix_bin have bugs in it.

A similar version is available called SMSQCommdLin_BIN which works in the same way, except for SMS SBASIC interpreters. Some versions of TurboFix_BIN also support SBASIC but it is currently recommended that this file is used instead.

**CROSS-REFERENCE**

See *TYPE_IN* for an example.

## 10.78 COMPILED

| | |
|---|---|
| Syntax | COMPILED |
| Location | Turbo Toolkit |

This function simply returns a value of 0 if the current program is interpreted or 1 if it has been compiled.

**NOTE 1**

Although primarily for use with programs compiled with Turbo, versions of this function after v3c27 will work even from within a program compiled under QLiberator.

**NOTE 2**

Prior to v3c27, this function did not always return the correct value on Minerva and SMS (particularly from within a MultiBASIC or SBASIC daughter job).

**CROSS-REFERENCE**

See *JOB_NAME* for an example.

## 10.79 COMPRESS

| | |
|---|---|
| Syntax | COMPRESS filename |
| Location | COMPICT |

This command takes the current screen contents and compresses them, saving the picture in its compressed form in the stated file - the full filename (eg. ram1_test_scr) has to be used.

This compressed form does not represent that great a saving over the original 32768 bytes required to hold the details of the screen before compression - the amount of space required for a compressed screen depends upon the amount of adjacent pixels on the screen which have the same colour.

Whilst the screen is compressed, a pattern is drawn over the screen, which although annoying, is harmless.

**Example**

COMPRESS flp2_TITLE_scr

**NOTE 1**

COMPRESS temporarily needs 64K of working space and will report an error if this is not available. Unfortunately the file stays open if this happens and cannot be accessed until is is closed with CLOSE% or a desktop program such as QPAC 2 (channels menu).

**NOTE 2**

COMPRESS does not work in supervisor mode, ie. it multitasks, thus if you were doing something else whilst the screen was being compressed, the saved picture may look pretty strange when expanded.

**NOTE 3**

COMPRESS assumes that the screen starts at $20000 and cannot therefore be used with Minerva's second screen or some emulator display modes.

**NOTE 4**

COMPRESS assumes a screen resolution of 512x256 and cannot work on higher resolution screens.

**CROSS-REFERENCE**

Screens which have been saved with *COMPRESS* can be loaded with *EXPAND* or re-loaded from memory with *FASTEXPAND*.

See also *SCR_STORE*.

## 10.80 CONCAT

| | |
|---|---|
| Syntax | CONCAT file1,file2 TO file3 |
| Location | CONCAT |

This command merges the first two files together to form a new file with the third specified filename, so that file2 is appended to file1. The length of file3 is exactly the sum of the lengths of the merged files.

**Example**

Most SuperBASIC programmers use their own standard set of

procedures and functions. If two of them need to be added

to a program, CONCAT helps a lot: CONCAT flp1_PROG_bas,flp1_SUB_1 TO ram1_PROG_tmp DELETE flp1_PROG_bas CONCAT ram1_PROG_tmp,flp1_SUB_2 TO flp1_PROG_bas DELETE ram1_PROG_tmp

You must ensure that line numbers do not conflict.

**NOTE**

Each filename must include the device.

**CROSS-REFERENCE**

*COPY*, *RENAME*, *DELETE*.

See *FWRITE* for the more flexible APPEND procedure.

## 10.81 CONNECT

| | |
|---|---|
| Syntax | CONNECT [#]pipe_in% TO [#]pipe_out% |
| Location | Turbo Toolkit |

This command is exactly the same as TCONNECT, except that the two channels do not have to have a hash sign in front of them.

**CROSS-REFERENCE**

*TCONNECT* and*QLINK*

## 10.82 CONTINUE

| | |
|---|---|
| Syntax | CONTINUE or<br>CONTINUE [line_no](Toolkit II & Minerva only) |
| Location | QL ROM, Toolkit II |

This command allows the user to try and recover from an error (normally after STOP or pressing the Break key), by telling the interpreter to carry on running the program from the next statement. This will however not work if the message 'PROC/FN Cleared'.

If you have Toolkit II, Minerva installed, you will be able to use the second variant of this command which allows you to re-start processing at a specified line number to help with error trapping.

**NOTE 1**

CONTINUE cannot carry on processing where the line which was stopped was a direct command (ie. typed in at #0).

**NOTE 2**

Unless you are using the Toolkit II or Minerva variants of this command, do not try to use CONTINUE after RENUMbering the program, as the continuation table is not updated by the RENUM routine and may therefore try to jump to the old line number.

**NOTE 3**

Beware that RENUM does not renumber line_no if you have used this command as part of a program.

**NOTE 4**

CONTINUE can only re-start processing if no new lines have been added; no new variables have been added to the program; no lines have been altered; and the PROC/FN Cleared message has not appeared.

**CROSS-REFERENCE**

See *RETRY* and also *WHEN ERRor*.

## 10.83 ConvCASE$

| | |
|---|---|
| Syntax | ConvCASE$ (string$ [,lower]) |
| Location | BTool |

ConvCASE$ returns the given string with all upper case letters converted to lower case if lower=1, or all lower case letters to upper case if lower=0. Default of lower is 1

**NOTE**

Unlike similar functions ConvCASE$ will recognise all non- ASCII letters, namely umlauts and accents.

**CROSS-REFERENCE**

*UPPER$*, *LOWER$*, *BIT%*, *CHR$*, *UPC$*, *LWC$*

## 10.84 CONVERT

| | |
|---|---|
| Syntax | CONVERT src_file,dst_file,original$,replacement$ |
| Location | CONVERT |

This command is used to copy src_file to dest_file and replace all occurrences of original$ by replacement$.

Both strings must have the same length.

The search is case-independent.

No default devices are supported.

**Example 1**

Take a QUILL-document and export it using the 'Print to file' option without a printer driver in the main drive.

Next VIEW it or look at it with an editor or by: COPY flp1_example_lis TO scr.

You will see the character CHR$(13) (the carriage return <CR> character) at the end of each line. This is not needed by QDOS to perform a carriage return on screen. Remove these excess characters with: CONVERT flp1_example_lis, flp1_example_txt, CHR$(13), " ".

<CR> at the end of lines may also appear when downloading messages from a bulletin board or converting MS/DOS text files to QDOS.

**Example 2**

Badly written or simple programs generally lack the feature to change device names for file operations. Using commands like FLP_USE may have a negative effect on any jobs which are running simultaneously, so it is better to make the program use flp1_ instead of mdv2_.

This can be achieved quite simply with the command: CONVERT prog1_exe, prog2_exe, "mdv2_", "flp1_".

**NOTE**

The character CHR$(0) cannot be replaced.

**CROSS-REFERENCE**

*EXCHG* is similar to *CONVERT*.

## 10.85 COPY

| | |
|---|---|
| Syntax | COPY file1 TO file2 or<br>COPY [file] [TO file2] (Toolkit II) or<br>COPY file1 [,file2 [,file3…]] {TO \| !} fileb (THOR XVI) |
| Location | QL ROM, Toolkit II, THOR XVI |

The command COPY duplicates file1, so that file2 is an exact copy. The parameters can also be a device (eg. ser1, con, scr, scr_400x20) or, if you have Toolkit II installed, a channel (eg. #3) can be used for the second parameter.

If Toolkit II is present, COPY supports the default devices and sub-directories. COPY will look for the file to be copied on the default data device if necessary (see DATAD$).

The rules for determining the destination parameter can be somewhat complex under Toolkit II:

(1) If no device is given, but a filename is specified, then Toolkit II looks at the first parameter. The destination device is then assumed to be the same as the source device (ie. the device name specified as part of the first parameter, or the default data device - see DATAD$).

Under SMS, it will use the default data device whether or not the first parameter contains a device.

(2) If the second parameter is omitted, then again Toolkit II looks at the first parameter. The same filename as for the first parameter will be used. If a device is given in the first parameter, then this is used as the destination device (unfortunately meaning that Toolkit II tries to copy the file onto itself!). On the other hand, if no device was specified, then the default destination device will be used (see DESTD$).

Under SMS, if a device is specified in the first parameter, SMSQ/E (v2.85 at least) tries to copy the file to the default destination device without a filename! Normally unless the default destination device is either PAR or SER, this will report an error 'is in use'.

(3) If a second parameter is given which includes a device name, then this is used!

If the destination is an existing file, unlike the normal ROM COPY command, Toolkit II will not break COPY with the error -8 (already exists), but instead it will print: >file<exists, OK to overwrite..Y or N? in channel #0 and wait for the user to press either <Y> or <N> - <ESC> and <CTRL><SPACE> mean <N> here.

**Examples**

Assuming that the default data device is flp1_ and the default destination device is ram2_ (using Toolkit II or SMS implementation):

| Command | Effect |
|---|---|
| COPY mdv1_quill TO flp1_quill | Copies mdv1_quill to flp1_quill |
| COPY ram1_prog_bas, ram2_tmp | Copies ram1_prog_bas to ram2_tmp |
| COPY ram1_prog_bas, scr_200x100 | Copies ram1_prog_bas to a window |
| COPY prog_bas, ser1 | Copies flp1_prog_bas to ser1 |
| COPY ser2 TO ram1_prog_bas | Copies data from ser2 to a file |
| COPY con TO ser | Copies everything typed to ser1 |
| COPY ram1_prog_bas | Tries to copy ram1_prog_bas to itself unless on SMS |
| COPY ram1_prog_bas | Tries to copy ram1_prog_bas to ram2_ and will report an error |
| COPY prog_bas | Copies flp1_prog_bas to ram2_prog_bas |
| COPY ram1_prog_bas TO #2 | Copies ram1_prog_bas to a channel |

**NOTE 1**

The TO separator can be replaced by a comma ',' (although note the THOR XVI variant!).

**NOTE 2**

Each file includes a file header of 64 bytes to store supple-mentary information such as the time of the last update, file type, length and much more. Without Toolkit II, COPY will always copy the header if a file is copied. The Toolkit II COPY command does not copy the header to serial devices (eg. ser) if this is specified as the destination.

**NOTE 3**

COPY without any parameters is allowed with Toolkit II, but it can cause problems (at least in versions up to v2.28 Toolkit II and v2.85 SMS)

As an exception to rule 2, when first used it would appear to try to copy the file "" on DATAD$ onto itself. On systems without level-2 drivers, such files can exist, but have no special function, whereas on level-2 drivers, these files contain the sub-directories. Thus, with the standard combination of Toolkit II and level-2 drivers installed, a pure COPY normally breaks with error -9 (in use) (see FMAKE_DIR for the reason).

However, due to a bug in current versions of Toolkit II, when first used it may report error -15 (bad parameter), in which case it will have left the file flp1_ open and prevent most of any further access to that device (unless you can close the channel with CLOSE% or a desktop).

**NOTE 4**

On SMS pre v2.58, if you used COPY and were asked if you wanted to overwrite the file, and answered N, an error code was returned.

**THOR XVI NOTES**

The THOR XVI (v6.41 and later) supports the third variant of COPY. This allows you to merge several files:

COPY flp1_texta,flp1_textb TO flp2_Book will create a new file flp2_Book made up of the merged files flp1_texta and flp1_textb. The headers will (of course) not be copied.

If you alter the TO delimiter to !, ie: COPY flp1_texta,flp1_textb ! flp2_Book

then it is assumed that fileb already exists and file1, file2 and file3 are all appended to it.

**CROSS-REFERENCE**

*SPL_USE* and *DEST_USE* set the destination device.

See *COPY_N* and *COPY_H* for copying file headers and *COPY_O* on how to force overwriting.

*SPL* performs a background copy (ie. it multitasks).

See *APPEND* which is similar to the THOR variant.

## 10.86 COPY_B

| | |
|---|---|
| Syntax | COPY_B adr1, adr2, n or<br>COPY_B adr1, n TO adr2 |
| Location | BTool |

The command COPY_B copies n bytes from the memory address adr1 to adr2 without any restrictions. The programmer has to ensure that there is sufficient room at the specified destination memory location (which must be free useable memory).

**Example**

```
100 RANDOMISE: n=10
110 a1=ALCHP(6*n): a2=ALCHP(6*n)
120 FOR i=0 TO 6*(n-1) STEP 6: POKE_F a1+i,RND
```

(continues on next page)

(continued from previous page)

```
130 COPY_B a1,6*n TO a2
140 FOR i=0 TO 6*(n-1) STEP 6: PRINT PEEK_F(a2+i)
150 RECHP a1: RECHP a2
```

**CROSS-REFERENCE**

*COPY_W*, *COPY_L*, *TTPOKEM* and *XCHANGE*

## 10.87 COPY_H

| | |
|---|---|
| Syntax | COPY_H [file1] [TO file2] |
| Location | Toolkit II |

See *COPY_N*.

**CROSS-REFERENCE**

*FGETH$*

## 10.88 COPY_L

| | |
|---|---|
| Syntax | COPY_L adr1, adr2, n or<br>COPY_L adr1, n TO adr2 |
| Location | BTool |

The command COPY_L copies n longwords (each being 4 bytes) from address adr1 to adr2. The two memory locations can overlap (this is also true for COPY_B and COPY_W).

If you are using Minerva, you will probably find it quicker to use it's specialised CALL routines.

**Example**

```
100 a=ALCHP(48*1024)
110 COPY_L 0,12*1024 TO a
```

**NOTE**

Both adr1 and adr2 must be even addresses.

**CROSS-REFERENCE**

*COPY_W*, *COPY_B*, *ODD*.

## 10.89 COPY_N

| | |
|---|---|
| Syntax | COPY_N file1 TO file2 or<br>COPY_N [file1] [TO file2] (Tooolkit II) or<br>COPY_N file1 [,file2 [,file3…]] {TO \| !} fileb (THOR XVI) |
| Location | QL ROM, Toolkit II, THOR XVI |

This command is basically the same as COPY, but the file header is explicitly removed. This is important for example if you wish to copy a file direct to a printer attached to ser2.

If the file header was also printed, this would include some non-printable characters {eg. CHR$(0)}, which might be interpreted by the printer as control characters and therefore produce rubbish as output.

Toolkit II's COPY examines the type of the destination device before it proceeds. It will not then copy the file header if this is a serial device or a parallel port. The standard COPY command contained in the QL ROM does not make this differentiation and so COPY_N must be used instead if the file header is not to be copied.

**CROSS-REFERENCE**

*COPY_H* forces the file header to be copied to the given destination (whether it is a serial port, a parallel port or not), and the syntax is identical to *COPY*, *COPY_N* and *COPY_O*.

## 10.90 COPY_O

| | |
|---|---|
| Syntax | COPY_O [file1] [TO file2] or<br>COPY_O file1 [,file2 [,file3…]] {TO \| !} fileb (THOR XVI) |
| Location | Toolkit II, THOR XVI |

The command COPY_O is identical to Toolkit II's COPY command, but if the file already exists, it will automatically be over-written without asking the user for confirmation.

This command is also supported on the THOR XVI, although both the input and destination channels must be specified in full.

**CROSS-REFERENCE**

*FTEST* and *ETAT* check the status of a file, thus enabling you to check if a file already exists.

## 10.91 COPY_W

| | |
|---|---|
| Syntax | COPY_W adr1, adr2, n or<br>COPY_W adr1, n TO adr2 |
| Location | BTool |

The command COPY_W copies n words (two bytes each) from address adr1 to adr2.

**NOTE**

Both addresses must be even.

**CROSS-REFERENCE**

*COPY_W* is always faster than *COPY_B*, but *COPY_L* is even faster than *COPY_W*.

See also *XCHANGE*.

Minerva has its own fast copy routines (see *CALL*).

## 10.92 COS

| | |
|---|---|
| Syntax | COS (radians) |
| Location | QL ROM |

This function allows you to find the cosine of the specified angle (given in radians).

In a right angled triangle the cosine is the ratio of the length of the side adjoining the given angle, to the length of the hypotenuse (or the sine of the complement of that angle). Thus, sine and cosine can actually substitute each other:

| Mathematical formula | In SuperBASIC |
|---|---|
| cos x = sin (PI/2-x) | COS(x)=SIN(PI/2-x) |
| sin x = cos (PI/2-x) | SIN(x)=COS(PI/2-x) |

**Example**

An analogue clock:

```
100 wx = 50: wy = INT(wx / 1.25): px = 50: py = 40
110 OPEN#3,"scr_" & wx & "x" & wy & "a" & px & "x" & py
120 PAPER#3,3: CLS#3: BORDER#3,1,0: SCALE#3,100,-45,-50
130 INK#3,0: FILL#3,1: CIRCLE#3,0,0,40:FILL#3,0: INK#3,4
135 Hs = PI/6
140 FOR t = 1 TO 12
150  LINE#3,40 * SIN(Hs * t), 40 * COS(Hs * t) TO 45 * SIN(Hs * t), 45 *␣
↪COS(Hs * t)
160 END FOR t
170 INK #3, 7
180 d$ = DATE$: min = d$(16 TO 17)
190 hour = d$(13 TO 14) MOD 12 + min / 60
200 LINE#3,0,0 TO 30 * SIN(Hs * hour), 30 * COS(Hs * hour)
210 LINE#3,0,0 TO 40 * SIN(PI / 30 * min), 40 * COS(PI / 30 * min)
220 PAUSE 100: CLOSE #3
```

**NOTE 1**

COS with very large values for the angle produces either very odd results or an overflow error (except on Minerva v1.96+ where it returns 0). The correct range for radians is -PI…PI, because anything outside this range is actually merely a repeat of the series. This is because an angle of PI*2 radians forms a complete circle, therefore an angle of PI*3 is actually the same as an angle of PI (ie. PI*3-PI*2). If you insist on using these silly angles, try SIN (X+PI/2) instead of COS(X).

**NOTE 2**

The THOR XVI (v6.41) fixes a slight inaccuracy in this command to ensure that COS(PI/2)=0. On other ROMs COS(PI/2)==0.

The Lightning package and SMS also fix this bug.

**CROSS-REFERENCE**

See *SIN*, *ACOS*.

Compare *COSH*.

Please also see the Mathematics section of the Appendix.

## 10.93 COSH

| | |
|---|---|
| Syntax | COSH (x) |
| Location | Hyper, Hyperbola |

This function is defined very similarly to SINH. It can be expressed as:

( EXP(x) + EXP(-x) ) / 2

**Example**

The COSH function can be used to describe a rope, chain or similar object which has two ends tied at the same height to a ceiling (for instance). Line 110 draws the ceiling, lines 120 to 160 the chain.

```
100 a = .8: SCALE 10, -5, 0: CLS
110 LINE -2,CHAIN(-2) TO 2,CHAIN(2)
120 FOR x = -2 TO 2 STEP .1
130   y = CHAIN(x)
140   IF x > -2 THEN LINE _x, _y TO x, y
150   _x = x: _y = y
160 END FOR x
170 :
180 DEFine FuNction CHAIN(x)
190  RETurn a * COSH(x/a)
200 END DEFine CHAIN
```

**CROSS-REFERENCE**

See *SINH* for an example.

*ARCOSH* is the inverse function of *COSH*.

## 10.94 COT

| | |
|---|---|
| Syntax | COT (angle) |
| Location | QL ROM |

This function returns the cotangent of a given angle (specified in radians).

In a right angled triangle the cotangent of an angle is defined as the ratio of the side adjoining the given angle to the side opposite to the given angle (forming a right angle with the other line). Due to the periodic nature of the function, it is best to work with angle in the range: 0 < angle < PI.

COT(angle) can also be calculated as COS(angle)/SIN(angle).

**Example**

A program to create a graph showing the range of the function COT:

```
100 MODE 4: OPEN#1,con_448x200a32x16
110 SCALE 100,-75,-50
120 INK 4: LINE -75,0 TO 125,0: LINE 0,-50 TO 0,50
130 CURSOR 0,0,0,0: PRINT '0'
140 CURSOR 0,0,-100,0: PRINT '-ã'
150 CURSOR 0,0,100,0: PRINT 'ã'
160 CURSOR 0,0,-200,0: PRINT '-ã * 2'
170 CURSOR 0,0,200,0: PRINT '2ã'
180 CURSOR 0,0,0,-100: PRINT '1'
190 CURSOR 0,0,0,90: PRINT '-1'
200 INK 7
210 FOR ang=-(PI*2)+1E-2 TO PI*2 STEP 1E-2
220   POINT ang*75/(PI*2),50*COT(ang)
230 END FOR ang
```

**NOTE 1**

Although COT(PI) and COT(PI*x) should be undefined (values of angle very close to PI tend to infinity), on most QDOS implementations, it gives a very large positive or negative number.

Currently, only the Lightning maths package and SMS produce an overflow error (the correct result).

**NOTE 2**

On Minerva v1.96+ very large values of angle will return the value 0. On other implementations produce an overflow error.

**NOTE 3**

COT(0) on most ROMs gives 1 - this is fixed on Minerva, SMS, Lightning, QXL, and ST/QL which give an overflow error.

**NOTE 4**

COT(PI/2) should equal zero - on all implementations of this command, this returns a number near to zero (except under SMS).

**CROSS-REFERENCE**

Please refer to*ACOT*, *ATAN*, *TAN*.

Compare *COTH*.

Also refer to the Mathematics section of the Appendix.

## 10.95 COTH

| | |
|---|---|
| Syntax | COTH (x) |
| Location | Hyper |

This function returns the hyperbolic co-tangent.

This is defined as one divided by the hyperbolic tangent, so COTH(x) = 1/TANH(x).

**CROSS-REFERENCE**

*ARCOTH* is the inverse function of *COTH*.

## 10.96 CSIZE

| | |
|---|---|
| Syntax | CSIZE [#channel,] width, height |
| Location | QL ROM |

This command sets the size and spacing of characters in the given channel (default #1).

Width ranges from 0 to 3 and there are two possible heights, 0 and 1. Each width and height corresponds to a certain pixel size:

| Width | Spacing | Size | Height | Spacing | Size |
|---|---|---|---|---|---|
| 0 | 6 | 5 | 0 | 10 | 9 |
| 1 | 8 | 5 | 1 | 20 | 18 |
| 2 | 12 | 10 | | | |
| 3 | 16 | 10 | | | |

In low resolution mode width 0 and 1 have no effect: in that mode, the smallest character size allowable is 12 pixels wide; CSIZE 2,0.

**NOTE 1**

On pre-JS ROMs, characters which use all eight pixels available for the definition of characters will not be printed correctly on screen. Even on JS and MG ROMs, problems exist in some character sizes. Minerva, SMS and the ST/QL drivers (Level E-23 onwards) prevent any such problems.

**NOTE 2**

The THOR XVI allows you to use any value for the vertical size - odd values give double height characters and even values give normal height.

**CROSS-REFERENCE**

*CHAR_INC* allows you to change spacing independently of character size.

*MODE* will reset the character size to the default (ie. 2,0 in *MODE* 8 and 0,0 in *MODE* 4).

The command *AT* is also affected by the current character spacing.

## 10.97 CTAB$

| | |
|---|---|
| Syntax | CTAB$ (string$ [,tabdist]) tabdist=1..255 |
| Location | BTool |

CTAB$ is a function which will look for spaces in the supplied string$ and if there is at least a tabdist number of spaces, they will be replaced by the TAB character, CHR$(9), so that ETAB$ or editors / word-processors can re-expand them to the original string.

CTAB$ does not alter the actual string$ but will return it in its compressed form.

The default value of tabdist is 8, and the length of string$ is limited to 255 characters (so tabdist>255 does not make much sense).

**WARNING**

tabdist=0 will produce rubbish output and it is also possible that CTAB$ will crash the system. Negative values lead to nonsense results but do not harm the system.

**CROSS-REFERENCE**

*ETAB$* expands the TAB marks.

## 10.98 CUR

| | |
|---|---|
| Syntax | CUR [#channel,] boolean |
| Location | TinyToolkit |

Every Window channel has a cursor which flashes when it is switched on and appears solid when it is inactive.

The command CUR with boolean=1 activates the cursor of a window, and it is de-activated with boolean=0. The default channel is #1.

**Example**

Multitasking programs should use INKEY$ to read keystrokes from the keyboard if no other job is to be similarly affected by the keys pressed. KEYROW could be used, but this does not care which job/channel/window was active when a key was pressed (this could be used to give a background job a command without leaving the current job).

The following function imitates the getchar() function of the C language, and is used for non-interactive keyboard input. Arcade games should not engage the cursor!

```
100 DEFine FuNction GETCHAR% (channel,timeout)
110  LOCal char$
120  CUR#channel,1
130  char$=INKEY$(#channel,timeout)
140  CUR#channel,0
150  RETurn CODE(char$)
160 END DEFine GETCHAR%
```

**NOTE**

Although the cursor was activated, it will not flash until the channel is made into the current keyboard input queue (ie. when it can use PEND, EOF, INKEY$). This may therefore mean that the keys <CTRL><C> will need to be pressed to make this program the active (current) task.

Non-console windows (scr_) cannot be used for input operations (ie. INPUT and INKEY$ cannot be used), nevertheless, the cursor may still be enabled.

INPUT will activate and de-activate the cursor itself.

**CROSS-REFERENCE**

See *FORCE_TYPE* concerning current input queue activation, *INKEY$*, *INPUT* and *KEYROW* for general information.

*CURSEN* and *CURDIS* are both combined by the *CUR* command. *CURSOR%*.

## 10.99 CURDIS

| | |
|---|---|
| Syntax | CURDIS [#ch] |
| Location | Toolkit II, THOR XVI, QSOUND |

This command disables the cursor in the given channel. See CURSEN!

The default window for this command is #1.

If a cursor is disabled in a given window (or does not exist), task switching with <CTRL><C> to the job which owns that window will not work unless the Pointer Environment is present.

**WARNING**

Do not use CURDIS #0 as this may prevent further input.

**CROSS-REFERENCE**

See *CURSEN* for more details.

*CURSOR_OFF* is similar.

## 10.100 CURSEN

| | |
|---|---|
| Syntax | CURSEN [#ch] |
| Location | Toolkit II, THOR XVI, QSOUND |

If a program is to multitask without the assistance of the Pointer Interface, it is necessary to give that program an active cursor so that the user can switch to the program using the key <CTRL><C>, which alters the active keyboard queue.

Unless a program has an active cursor, it cannot accept input from the keyboard by the use of commands such as PAUSE, INKEY$ and INPUT.

The command CURSEN enables the cursor in the given channel, which must be either a scr or con channel. If no channel is specified, the default is #1.

Once the cursor is enabled, a red block will appear at the current text cursor position in the given channel. This block will begin to flash when the cursor is 'active' (ie. expecting input).

**MINERVA NOTE**

Minerva's System Xtensions allow you to alter the attributes of the text cursor, by using the command POKE !124!51,x where x is in the format of: RRRRSCCC, where the top 4 bits of x (RRRR) determine the cursor flash rate, the bottom three bits (CCC) determine the colour of the cursor and the fourth bit (S) determines whether the cursor appears as a solid block or an underline.

You can actually get an invisible cursor by using the command POKE !124!51,0. Unfortunately though, this sets the cursor attributes for all cursors which are enabled, rather than just for the current Job.

**CROSS-REFERENCE**

*KEYROW* reads keys without an active cursor. See *CURDIS* also.

## 10.101 CURSOR

| | |
|---|---|
| Syntax | CURSOR [#channel,] [grx, gry,] x,y or<br>CURSOR [#channel,] flag (Btool only) |
| Location | QL ROM, Btool |

The CURSOR command allows you to set the text cursor to a specific position in the given window (default #1). Any text which is then printed will appear with the given position at its top left corner.

The values x and y specify the position in pixel co-ordinates relative to the origin of the specified window (eg. if the window #1 was defined as scr_448x200a32x16, the command CURSOR 224,100 will set the text position to the exact centre of the window).

However, for the more adventurous, CURSOR can take an additional two parameters which allow you to mix text and graphics on a given window more easily. This sets the text cursor to the graphics co-ordinate (grx,gry) and then uses the x and y parameters to specify a relative pixel offset from this graphics co-ordinate (a positive value of x moves the text cursor to the right, a negative value to the left; whereas a positive value of y moves the text cursor down, a negative value up).

The second variant only works with the Btool Toolkit. This allows you to enable or disable the cursor in the specified window (default #1), by specifying a flag of 1 to enable the cursor or 0 to disable the cursor.

**Example**

This program shows all the 45 degree angles in a circle:

```
100 MODE 4:WINDOW 448,200,32,16
110 PAPER 0:INK 7:CLS
120 SCALE 200,-150,-100
130 FOR i=0 TO 315 STEP 45
140   INK 7:LINE 0,0 TO SIN(RAD(i))*50,COS(RAD(i))*50
150   xoff=0:yoff=0
160   SELect ON i
170     =0:xoff=-4:yoff=-9
180     =45:yoff=-9
190     =90:yoff=-4
200     =180:xoff=-10
210     =225:xoff=-20
220     =270:xoff=-20:yoff=-4
```

(continues on next page)

(continued from previous page)

```
230     =315:xoff=-20:yoff=-9
240   END SELect
250   INK 4:CURSOR SIN(RAD(i))*50,COS(RAD(i))*50,xoff,yoff
260   PRINT i
270 END FOR i
```

**NOTE 1**

On pre MG ROMs, the CURSOR command only allows a maximum of four parameters, which means that you can only use grx, gry, x and y on the default channel. You can however use commands such as CURSOR #3,200,40,3 - although this is not supported on Minerva (pre v1.98) and SMSQ/E and should be avoided! If you specify a fifth parameter, a 'Bad Parameter' error will be reported. The Btool variant fixes this as does SMS, MG ROMs and Minerva. Compiling with Q-Liberator does not prevent this error.

**NOTE 2**

The graphics positioning did not work on ST/QL Emulators with Drivers prior to Level D-15 (or E-15).

**NOTE 3**

Compilers will not accept the second syntax.

**NOTE 4**

As from SMS v2.74 CURSOR limits grx to even positions to make in compatible with MODE 8 and MODE 4 automatically.

**CROSS-REFERENCE**

Please refer to *PRINT*, *LEFT* and *AT*.

*CURSEN* is a more compatible means of enabling a cursor.

## 10.102 CURSOR%

| | |
|---|---|
| Syntax | CURSOR% [#window] |
| Location | BTool |

This function returns the current status of the text cursor in the specified window (default #1). Results are:

- 0 for disabled,
- 1 for enabled and visible,
- -1 for enabled but invisible.

An active cursor flashes, and therefore alternates between visible and invisible status when enabled. Otherwise, it will appear as a solid block on screen (unless there is no cursor attached to the specified channel).

On Minerva it is possible to alter the shape and colour of the cursor.

**CROSS-REFERENCE**

*CURSEN*, *CURDIS*, *CURSOR* and *CUR* enable or disable the cursor.

Also refer to *CURSOR_OFF* and *CURSOR_ON*.

## 10.103 CURSOR_OFF

| | |
|---|---|
| Syntax | CURSOR_OFF [#ch] |
| Location | Turbo Toolkit |

This command is exactly the same as *CURDIS*.

## 10.104 CURSOR_ON

| | |
|---|---|
| Syntax | CURSOR_ON [#ch [!] ] |
| Location | Turbo Toolkit |

This command is very similar to *CURSEN*, with the default window being #1.

However, you can add an exclamation mark after the channel number. If this is omitted, then upon execution of this command the chosen window is automatically selected as the active window (where key input is directed). Add the exclamation mark to prevent this.

**CROSS-REFERENCE**

See *CURSOR_OFF*, *CURSEN* and *CURSOR%* for more details.

## 10.105 CVF

| | |
|---|---|
| Syntax | CVF (mkf_$) |
| Location | BTool |

This function takes any six character long string, (the internal format of a floating point number), and returns the value as a floating point number.

**WARNING**

CVF locks SuperBASIC if the supplied parameter is six bytes long but not a valid representation of a floating point number, eg. CVF("BlaBla").

MKF$ always returns a valid parameter for CVF which will not crash it.

**CROSS-REFERENCE**

*PEEK_F*, *MKF$*, *CVI%*, *CVL*, *CVS$*.

*FPUTF* and *FGETF* enable you to read and write floating point numbers in internal format to or from files.

## 10.106 CVI%

| | |
|---|---|
| Syntax | CVI% (mki_$) |
| Location | BTool |

CVI% is the inverse function of *MKI$* and expects a two character long string, being the internal representation of an integer, and then converts this into the actual integer number.

**Example**

MKI$( 20812 )="QL"

CVI%("QL")=20812

**CROSS-REFERENCE**

*MKI$*, *CVL*, *CVF*, *CVS$*.

*FGET%* and *FPUT%* provide similar facilities for writing and reading integers in their internal format from files.

## 10.107 CVS$

| | |
|---|---|
| Syntax | CVS$ (mks_$) |
| Location | BTool |

This function takes the internal representation of a string and returns the string concerned.

A string is represented internally as a word containing the length of the string followed by the string itself.

**Example**

CVS$( CHR$(0) & CHR$(2) & "Test" ) = "Test"(1 TO 2) = "Te"

**CROSS-REFERENCE**

*MKS$*, *CVI%*, *CVL*, *CVF*. *FPUT$* and *FGET$* enable you to write strings to and read strings from files in their internal formats.

## 10.108 CVL

| | |
|---|---|
| Syntax | CVL (mkl_$) |
| Location | BTool |

This function converts the internal representation of a long integer number (a four character long string) to the actual value and returns that. CVL is the inverse of *MKL$*.

**Example**

CVL( MKL$( 10010 ) = "10010"

**CROSS-REFERENCE**

*MKL$*, *CVI%*, *CVF*, *CVS$*.

*FPUTL* and *FGETL* provide similar facilities to enable you to write and read long integers from files in their internal format.

CHAPTER

# ELEVEN

# KEYWORDS D

## 11.1 DATA

| | |
|---|---|
| Syntax | DATA expression *[,expression]* |
| Location | QL ROM |

The QL allows a SuperBASIC program to store a set of data in the program itself, which can then be assigned to a given variable by the READ command. The DATA statement marks these areas for use by READ. The information which can be stored at a DATA statement is basically anything which can be stored in a variable, including strings, variables, constants and expressions.

Expressions will be calculated at the time that the item in question is READ. Whilst a program is running, unless a READ command is found, DATA statements are ignored.

**Example**

```
1000 DATA "QL User",100,x*1000+10
```

**NOTE 1**

On Pre MG ROMs, if any values in a DATA statement start with a bracket, then the other items on the line may be ignored. If you must specify items starting with brackets, use for example: DATA 0+(... This is fixed by MG ROMs, Minerva and SMS.

**NOTE 2**

Unless you have a Minerva ROM (v1.77 or later) or SMS, when you enter the DATA statement, you will always need to type a space after the word DATA as the parser will not automatically insert one. On later implementations a space is automatically inserted where the first DATA expression is a string, eg. DATA'Hello'.

**NOTE 3**

Entering a DATA statement as a direct command from #0 has no effect. Under SMS an error is reported 'DATA in command line has no meaning'.

**NOTE 4**

Due to the way in which the interpreter works, it is always more efficient to place DATA statements at the start of a program (the search function always starts at the first line of the program).

**NOTE 5**

Various SuperBASIC compilers (such as Turbo) do not support expressions in DATA statements.

**NOTE 6**

There appears to be no real check on the parameters given for DATA, so the following line can be entered, but will in fact cause an error when you try to READ it:

```
10 DATA 1000,PRINT,10
```

SMS's improved interpreter does do more checks than earlier implementations and will prevent you from entering the line:

```
10 DATA 1,1,2a,3
```

which other implementations allow (but give an error when they try to READ the line).

**NOTE 7**

SMS may complain if you create numerous DATA statements inside a DEFine PROCedure or DEFine FuNction struture.

**CROSS-REFERENCE**

*RESTORE* allows you to set the current *DATA* pointer. *READ* will assign the value at the current *DATA* pointer to the given variable. *EOF* will return the value one if there are no more *DATA* statements in the current program.

## 11.2 DATAD$

| | |
|---|---|
| Syntax | DATAD$ |
| Location | Toolkit II |

This function always contains the current default data device, which is an unofficial QDOS standard and supported by all Toolkit II extensions, original SuperBASIC commands and most good software.

The default device means that if no other device is stated, if appropriate, this device will be used. The default data device will also be consulted if a device name is supplied but the given file cannot be found on that device. For example, assuming that DATAD$='flp2_', if you enter VIEW ram1_example_txt and the file example_txt is not present on ram1_, the command will then try flp2_ram1_example_txt.

This idea can be extended to use prefixes as sub-directories. Sub-directories are separated by underscores, DATAD$ always ends with an underscore.

**Example**

TK2DIR reads all files from the current default data device via a pipe, strips off any network sub-directory prefix and then writes the remainder of the filenames into the string array passed by parameter.

```
100 DEFine PROCedure TK2DIR (Verz$)
110   LOCal e,n,sd$,sd,us
120   sd$=DATAD$: us="_" INSTR sd$
130   IF us=3 AND LEN(sd$)>3 and sd$(1)="n" THEN
140     IF sd$(2) INSTR "12345678":sd$=sd$(4 TO):us="_" INSTR sd$
160   END IF
170   OPEN#4,pipe_10000: STAT#4: WDIR#4
180   e=FILE_OPEN(#3,pipe_,CHANID(#4)): CLOSE#4
```

(continues on next page)

(continued from previous page)

```
200   INPUT#3,Verz$(0)
210   FOR n=1 TO DIMN(Verz$)
220     IF EOF(#3) THEN EXIT n
230     INPUT#3,Verz$(n)
240     Verz$(n)=Verz$(n)(us+1 TO)
250   END FOR n
260   CLOSE#3
270 END DEFine TK2DIR

DIM file$(20,30)
TK2DIR file$
CLS: PRINT file$
```

Here only the first 20 files will be read into file$. NB. This would require substantial amendment to make it search sub-directories also.

**CROSS-REFERENCE**

*DATA_USE* defines the default device; *DUP*, *DDOWN* and *DNEXT* allow you to move around the sub-directory tree. *PROGD$* returns the default program device. *DLIST* prints all default devices.

## 11.3 DATAREG

| | |
|---|---|
| Syntax | DATAREG [number]number=0…3 |
| Location | TRAPS (DIY Toolkit Vol T) |

This function returns the value of the Machine code data register number (default 0) following the completion of a MTRAP, QTRAP or BTRAP command.

Because the default data register number is 0: PRINT DATAREG will be 0 if no error occured during the TRAP call or else the relevant error code.

Number will let you read the value of the relevant data register D0, D1, D2 or D3.

**CROSS-REFERENCE**

*ADDREG* allows you to read machine code address registers - see this for an example of *DATAREG*. See *MTRAP*, *QTRAP* and *BTRAP*.

## 11.4 DATASPACE

| | |
|---|---|
| Syntax | DATASPACE (file$) |
| Location | Turbo Toolkit |

This function returns the amount of dataspace which has been set aside for the given file$. It is therefore similar to FDAT and FILE_DAT.

Default devices are not supported, however errors are not reported. The following error values may also be returned by the function:

- -2: The file is not executable

- -3 or -6: Insufficient memory to open file
- -7: File does not exist
- -9: Device or file is being written to by something else.
- -12: The device is valid, but the filename is not
- -16: Bad or changed medium error

**Example**

```
PRINT DATASPACE('win1_start_QD_exe')
```

**CROSS-REFERENCE**

*DATA_AREA* allows you to set the dataspace for a compiled program. See also *FDAT*.

## 11.5 DATA_AREA

| | |
|---|---|
| Syntax | DATA_AREA size size=0…850 |
| Location | Turbo Toolkit |

This command is only used by the Turbo compiler and should be located at the start of your program before any active program lines.

The command specifies how much dataspace (size kilobytes) should be specified for the compiled program.

This dataspace is used by a task for stack space and a temporary store whilst it is running.

**Example**

```
10 DATA_AREA 32
```

**NOTE**

This setting will override a previous TURBO_objdat directive in the same program. It will also be overridden by a later TURBO_objdat directive in the same program.

**CROSS-REFERENCE**

*DATASPACE* allows you to find out how much dataspace has been set aside for a program. See *COMPILED* and *TURBO_objfil* for other compiler directives. *TURBO_objdat* is exactly the same.

## 11.6 DATA_USE

| | |
|---|---|
| Syntax | DATA_USE default_device |
| Location | Toolkit II, THOR XVI |

If you have Toolkit II installed, all of the additional extensions connected with file or device handling and all original SuperBASIC commands use the default device if no other device name is specified.

On a THOR XVI, some of the commands support default devices without Toolkit II.

The effect of the default devices would make LOAD proggy_bas work as LOAD flp1_proggy_bas (assuming that flp1_ is the default data device). The actual effect depends on the command being executed, but generally the file will be looked for in three steps:

- Does the given file include a valid device? proggy_bas does not, ram1_proggy_bas does (ram1_). If not, the parameter is assumed to be a filename and Toolkit II looks for a device on which it can find it; so:
- Add the default data device to the filename. If that does not work, then:
- Add the default program device (PROGD$) and try again.

The default program device is defined by PROG_USE, DATA_USE defines the default data device. See PROG_USE as to the difference between the two defaults. The last two steps add the default devices to the filename. These defaults can be interpreted as sub-directories.

Here, a sub-directory means that where a prefix is separated by underscores, this means that the file concerned is held in the sub-directory specified by that prefix. Thus, win1_QUILL_readme_doc could be readme_doc on a hard disk in the sub-directory QUILL or doc in the sub-subdirectory readme of QUILL.

Sub-directories can be nested but the complete filename, including prefix must not be longer than 41 characters (note that if you are using a network device, for example n1_win1_proggy_bas, the maximum permitted filename length is reduced to 39 in current versions of the QL device drivers).

**Examples**

```
DATA_USE flp1_QUILL (or flp1_QUILL_)
DATA_USE MDV2_
DATA_USE win1_Psion_ARCHIVE
DATA_USE n2_ram1_
DATA_USE mdv3_games_arcade_invaders_
```

**NOTE 1**

If there is no underscore at the end of DATA_USE's parameter, it will be added automatically.

**NOTE 2**

A few programs do work with these sub-directories (if Toolkit II is present), but most do not. To make any program work with them, you can fool them so that they believe that for instance FLP1_games_BOOT is FLP1_BOOT or BOOT (default device FLP1_games): See the PTH_… and DEV_… commands.

**NOTE 3**

Toolkit II sub-directories should not be mixed up with wild cards. DATA_USE flp1__bas makes WDIR list all BASIC programs on floppy 1, but after PROG_USE flp1__bas, SAVE test will not save the current program as flp1_test_bas but as flp1__bas_test.

**NOTE 4**

The default device is the current sub-directory on level-2 drivers.

**NOTE 5**

If you wish to turn off this feature, you can assign a null string ("") to DATA_USE.

**NOTE 6**

The default devices cannot exceed 32 characters (plus a final underscore) - any attempt to assign a longer string will result in the error 'Bad Parameter' (error -15).

**CROSS-REFERENCE**

*DATAD$* contains the default data device, *DLIST* lists all default devices. *DDOWN*, *DUP* and *DNEXT* allow you to skip from sub-directory to sub-directory, climb up the tree and much more. *PROG_USE* changes the default program device, and *SPL_USE* /*DEST_USE* the default destination device. See also *DEV_USE* and *PTH_ADD* for path search.

## 11.7 DATE

| | |
|---|---|
| Syntax | DATE or<br>DATE (year,month,day,hour,minute,second)(Minerva & NewDate) or<br>DATE (year,month,day,hour,minute [,second])(SMS v2.57+) |
| Location | QL ROM |

The function DATE returns the current date and time as the number of seconds since midnight on 1st January 1961. For example, PRINT DATE$(DATE) is exactly the same as PRINT DATE$. The NewDate version of this command is exactly the same as Minerva's implementation.

**NOTE**

Due to the way in which the system clock is implemented on the QL (it is stored as a 32-bit unsigned number), early versions of this function have problems with dates after 3.14:07 on 19th January 2029 (this would result in a number of seconds which needs to be stored in all 32 bits).

Although the SDATE and DATE$ functions treat the number correctly, the DATE function ignores the most significant bit, meaning that it returns the wrong value for dates later than this.

The NewDate version of this function, as well as Minerva ROMs and under SMS, DATE treats the figure as a 32-bit signed number. Although this allows the line PRINT DATE$(DATE) to work correctly for all dates between 0.0:00 on 1st Jan 1961 and 6.28:15 on 6th Feb 2097, note that any dates after 3.14:07 on 19th January 2029 are returned as negative numbers, with earlier dates giving the largest negative number.

**MINERVA NOTE**

DATE can accept the same six parameters accepted by SDATE. This enables you (for instance) to find out the day on a given date without having to alter the QL clock: PRINT DAY$(DATE(1968,6,25,1,1,0))

This does also enable you to easily set the update date on a given file without altering the QL clock:

```
SET_FUPDT \flp2_test_file, DATE(1990,11,1,0,0,0)
```

**SMS NOTE**

As from v2.57, DATE has been brought up to the same standard as on Minerva. However, the seconds do not have to be specified and will default to zero if omitted.

**CROSS-REFERENCE**

*SDATE* will alter the QL clock. *DAY$* returns the day on the given date, *DATE$* will return the current date. *T_ON* and *T_START* can be used for accurate stop-watches for timing programs.

## 11.8 DATE$

| | |
|---|---|
| Syntax | DATE$ [(date)] or<br>DATE$ (year,month,day,hour,minute [,second])(SMS v2.57+ only) |
| Location | QL ROM |

DATE$ holds the current system date and time as a string in the following format: yyyy mmm dd hh:mm:ss.

```
1991 May 06 18:18:44 (example)
|   | | | || || || ||
|   | | | || || || ++---- 19 TO 21 (seconds)
|   | | | || || ++------- 16 TO 17 (minutes)
|   | | | || ++---------- 13 TO 14 (hour, 24h)
|   | | | ++------------- 10 TO 12 (day)
|   | +-+----------------- 6 TO 8 (month as string)
+--+--------------------- 1 TO 4 (year)
```

If a parameter is used then DATE$ should return the date and time the given number of seconds after 1/1/1961, DATE$(DATE) is identical to DATE$ for any date before 3.14:07 on 19th Jan 2029 (see ADATE). However, for times after this date, the number of seconds since 1/1/1961 is represented by a negative number, calculated by number of seconds - 2147483648.

This means that to calculate a specified date after 3.14:06 on 19th Jan 2029, the following short function is required (for non-Minerva ROMs and non-SMS machines only):

```
100 DEFine FuNction DATE20$(seconds)
110   offset='2147483648'
120   RETurn DATE$(seconds-offset)
130 END DEFine
```

This function is not needed on Minerva ROMs, with the NewDate version of DATE or under SMS - see DATE for a full explanation.

**Example 1**

It may be useful to read the different parts of the date from DATE$ and reformat them for use in letters.

```
100 D$=DATE$
110 year=D$(1 TO 4): day=D$(10 TO 12): D$=D$(6 TO 8)
120 month=(D$ INSTR "..JanFebMarAprMayJunJulAugSepOctNovDec")/3
130 DIM month$(12,9): RESTORE 150
140 FOR m=1 TO 12: READ month$(m)
150 DATA "January","February","March","April","May","June","July"
160 DATA "August","September","October","November","December"
170 ALTKEY "d", month$(month) & " " & day & ", " & year
```

**Example 2**

How to find the number of days between two dates:

```
100 date1=DATE(2032,3,30,10,0,0)
110 date2=DATE(2000,3,30,10,0,0)
120 PRINT DAYS_DIFF(date2,date1)
130 :
140 DEFine FuNction DAYS_DIFF(dy1,dy2)
150 LOCal offset,base_date,diff
160 offset='2147483648'
170 base_date=DATE(2029,1,19,3,14,7)
180 IF (date1>=0 AND date2>=0) OR (date1<0 AND date2<0)
190 IF date1>=date2:diff=date1-date2:ELSE diff=date2-date1
240 ELSE
250 IF date1<0
260 diff=(base_date-date2)+(date1+offset
270 ELSE
280 diff=(base_date-date1)+(date2+offset)
290 END IF
300 END IF
310 seconds_per_day=24*60*60
320 RETurn INT(diff/seconds_per_day)
330 END DEFine
```

**NOTE 1**

Parts of string functions cannot be obtained by slicing them directly. Expressions such as DATE$(DATE)(1 TO 4) are only valid on Minerva ROMs or under SMS. On other ROMs, the value of the function has to be copied to a variable before being sliced (as demonstrated in example 1).

**NOTE 2**

The QL's system clock is limited in the range of dates it can cover - see ADATE.

**MINERVA NOTE**

Although on Minerva (v1.77 and later), DATE$ can now be directly sliced to extract the year for instance. It is however, necessary to tell the operating system that you are not actually providing a parameter to be converted into a date. This is achieved by using the following format to slice DATE$: DATE$ [([seconds]) [([start] TO [end])]] The following are therefore all valid on Minerva:

```
PRINT DATE$
PRINT DATE$(DATE+86400)
TIMER$ = DATE$()(13 TO )
YEAR$ = (DATE$)(1 TO 4)
YEAR$ = DATE$(1E9)( TO 4)
```

Only the first two examples will work on other ROMs.

**SMS NOTE**

DATE$ works mainly as per Minerva, however from v2.57+, you can also supply five or six parameters to DATE$ in common with DATE and SDATE.

**CROSS-REFERENCE**

Use *SDATE* and *ADATE* to set and alter the system time and date. *DATE* holds the current date as a floating point number, *DAY$* holds the weekday as a short string.

## 11.9 DAY$

| | |
|---|---|
| Syntax | DAY$ [(date)] or<br>DAY$ (year,month,day,hour,minute [,second]) (SMS v2.57+ only) |
| Location | QL ROM |

DAY$ holds the current day as a three character string:

| | |
|---|---|
| Sun | Sunday |
| Mon | Monday |
| Tue | Tuesday |
| Wed | Wednesday |
| Thu | Thursday |
| Fri Sat | Friday Saturday |

If you provide a parameter, DAY$ will return the day of the given date (which is stated in seconds after 1/1/1961). DAY$(DATE) = DAY$.

**NOTE**

As with DATE$, you cannot slice DAY$ unless you have a Minerva ROM (version 1.77 or later) or SMS - see DATE$ for further details.

**SMS NOTE**

In common with DATE$, from v2.57, DAY$ will now accept five or six parameters as with SDATE and DATE. You can also slice DAY$ (like on Minerva) - see DATE$.

**CROSS-REFERENCE**

*TRA* and *SET_LANGUAGE* allow you to re-define the abbreviations used for the different days. *DATE* holds the current system date (in seconds after 1/1/1961) as a floating point number, *DATE$* as a string.

## 11.10 DAY%

| | |
|---|---|
| Syntax | DAY% [datestamp] |
| Location | SMSQ/E |

This function complements the *DATE* and *DATE$* functions, by returning the day number corresponding to the given datestamp, or current date, if no datestamp was given.

**Examples**

```
PRINT DAY% (0)
```

will print the day part of the QL's epoch, 1 for 1st of January

```
PRINT DAY%
```

will print the current day number.

**CROSS-REFERENCE**

See *DATE*, *YEAR%*, *MONTH%*.

## 11.11 DBL

| | |
|---|---|
| Syntax | DBL |
| Location | Beuletools |

This function returns the control codes needed to switch on emphasised mode on an EPSON compatible printer: DBL=CHR$(27)&"E".

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*, *ENL*, *PRO*, *SI*, *NRM*, *UNL*, *ALT*, *ESC*, *FF*, *LMAR*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 11.12 DDOWN

| | |
|---|---|
| Syntax | DDOWN subdirectory |
| Location | Toolkit II |

This command adds the specified subdirectory to the default data device as a suffix.

If the default program device is the same as the default data device, then this will also be altered by DDOWN.

If the default destination device is a directory device (ie. if it ends with an underscore), DDOWN also alters this (whether or not it points to another drive).

```
win1_
win1_C_
win1_C_include_
win1_C_objects_
win1_BASIC_
win1_QUILL_
win1_QUILL_letters_
win1_QUILL_translations
win1_secret_
```

The above could be a directory tree on a hard disk.

DATA_USE win1_ defines win1_ as the default directory device, so WDIR will list all of the files on win1_.

DDOWN C will move into the C sub-directory, ie. DATAD$ is now win1_C_.

DDOWN include will make WDIR list all of the files on the hard disk which are prefixed by C_include_ (eg. win1_C_include_math_h).

**NOTE 1**

DDOWN does not check if there are any files with the given prefix which exist.

**NOTE 2**

DDOWN breaks with error -17 (error in expression) if the parameter is a resident keyword. So append an underscore to the directory name, eg. DDOWN NEW_, or specify the parameter between quote marks (eg. DDOWN 'NEW').

**NOTE 3**

The default devices cannot exceed 32 characters (plus a final underscore) - any attempt to extend them beyond this will result in the error 'Bad Parameter' (error -15).

**CROSS-REFERENCE**

*DUP* moves up the tree, *DNEXT* skips from branch to branch. *DATAD$* and *DLIST* can be used to find out about the current sub-directory and default devices respectively.

## 11.13 DEALLOCATE

| | |
|---|---|
| Syntax | DEALLOCATE address |
| Location | Turbo Toolkit |

This procedure is very similar to RECHP in that it cancels a reservation of common heap memory. However, the specified address must be an area of memory which had previously been set aside with ALLOCATION.

**WARNING**

Prior to v3d27 this command could crash the system if the specified address had already been deallocated, was an odd address, or had not been set aside with ALLOCATION.

**CROSS-REFERENCE**

See *ALLOCATION* and *RECHP*.

## 11.14 DEBUG

| | |
|---|---|
| Syntax | DEBUG |
| Location | Turbo Toolkit (v3.20+) |

This is a compiler directive intended to precede a DEFine PROCedure or DEFine FuNction routine which is used for debugging a program. The routine can be included or excluded from the program during compilation using the DEBUG_LEVEL directive. Current versions of the TURBO parser do not support this.

**CROSS-REFERENCE**

See *DEBUG_LEVEL*.

## 11.15 DEBUG_LEVEL

| | |
|---|---|
| Syntax | DEBUG_LEVEL level |
| Location | Turbo Toolkit (v3.20+) |

It is currently uncertain how this directive is used within TURBO compiled programs.

**CROSS-REFERENCE**

See *DEBUG* and the various TURBO_XXX commands starting with *TURBO_diags*.

## 11.16 DEFAULT

| | |
|---|---|
| Syntax | DEFAULT (expression, default_value) |
| Location | BTool |

The function DEFAULT usually simply returns the result of the given expression, unless the expression contains undefined variables or does not produce a floating point number. In either of these latter cases DEFAULT will return the given default_value.

**Example**

WRITE simply PRINTs a text to a given channel. If the channel ch was not a valid number for any reason then #1 is used:

```
100 DEFine PROCedure WRITE (ch, text$)
110 ch = DEFAULT(ch, 1)
120 PRINT#ch,text$
130 END DEFine WRITE
```

**CROSS-REFERECE**

*TYPE*. *DEFAULT$* and *DEFAULT%* work exactly like *DEFAULT* for string and integer expressions.

## 11.17 DEFAULT%

| | |
|---|---|
| Syntax | DEFAULT% (expression%, default_value%) |
| Location | BTool |

**CROSS-REFERENCE**

See *DEFAULT* !

## 11.18 DEFAULT$

| | |
|---|---|
| Syntax | DEFAULT$ (expression$, default_value$) |
| Location | BTool |

**CROSS-REFERENCE**

See *DEFAULT* !

## 11.19 DEFAULT_DEVICE

| | |
|---|---|
| Syntax | DEFAULT_DEVICE devicename$ |
| Location | Turbo Toolkit |

This command can be used in a similar way to PROG_USE and DATA_USE. It sets the default device (up to 31 characters), for the following Turbo Toolkit commands:

- CHARGE,
- EXECUTE,
- EXECUTE_A,
- EXECUTE_W
- LINK_LOAD,
- LINK_LOAD_A,
- LINK_LOAD_W.

It has no effect on any other commands.

**Example**

For a series of linked programs, you may want to use the following in a boot file:

```
DEFAULT_DEVICE win1_PROGS_
```

Each program could call another by using:

```
EXECUTE_W program2_task
```

**NOTE 1**

Prior to v3d27, this command only supported 5 characters (although prior to v2.00 no error was reported if more than 5 characters were used - the command simply ignored the additional characters).

**NOTE 2**

As from v1.26, you do not need to pass the device name as a string, for example:

```
DEFAULT_DEVICE flp1_
```

**CROSS-REFERENCE**

*PROG_USE*.

## 11.20 DEFAULT_SCR

| | |
|---|---|
| Syntax | DEFAULT_SCR |
| Location | Fn (v1.02 or later) |

This function is really only useful on a Minerva ROM (although it will work quite happily on any other ROM). It is sometimes useful when writing programs which are to run in Minerva's dual screen mode to

discover which is the default screen. This is made necessary because all new windows which are opened, and all MODE commands operate on the current default screen.

This therefore means that if a program is badly written, it is possible that whilst the program is running the default screen is switched, giving the result that some of its windows are opened on scr0 and some on scr1. PRINT DEFAULT_SCR will return 0 or 1 depending whether the default screen is scr0 or scr1. If Minerva is not in dual screen mode, or if Minerva is not present, 0 will be returned.

**Example**

A program to change the MODE of the current program safely (ie. it will only alter the MODE of the screen in which the program is running):

```
100 This_JOB=DEFAULT_SCR
110 SET_MODE 8
120 :
200 DEFine PROCedure SET_MODE (alp)
210   IF RMODE(This_JOB)=alp:RETurn
220   IF This_JOB=DEFAULT_SCR:MODE alp:RETurn
230   MODE 64+32,-1:MODE alp:MODE 64+32,-1
240 END DEFine
```

**CROSS-REFERENCE**

*MODE* alters the mode of the current screen and job and can be used to alter the current default screen, *RMODE* returns the mode of the given screen.

## 11.21 DEFine xxx

| | |
|---|---|
| Syntax | DEFine .... |
| Location | QL ROM |

This keyword forms part of the structures:

- DEFine PROCedure,
- DEFine FuNction,
- END DEFine.

As such, it cannot be used on its own within a program - this will cause a 'bad line' error, except under SMS where it causes an error 'Incorrect Procedure or Function Definition'.

**CROSS-REFERENCE**

Please refer to the individual structure descriptions for more details: *DEFine FuNction*, *DEFine PROCedure* and *END DEFine*.

## 11.22 DEFine FuNction

| | |
|---|---|
| Syntax | DEFine FuNction name[$ \| %] [(item *[,item$^i$]* )] |
| Location | QL ROM |

This command marks the beginning of the SuperBASIC structure which is used to surround lines of SuperBASIC code which forms an equivalent to a machine code function, which can be called from within SuperBASIC and will return a value dependent upon the code contained within the structure. The syntax of the SuperBASIC structure can take two forms:

---

DEFine FuNction name[$ | %] [(item*[,item^i]*)]: statement *[:statement]*:RETurn value

or

DEFine FuNction name[$ | %] [(item *[,item^i]* )] *[LOCal var *[,var^i]*]* *[statements]* RETurn value
END DEFine [name]

---

When the specified function name is called, the interpreter will search the SuperBASIC program for the related DEFine FuNction statement.

If a related DEFine FuNction cannot be found, then the interpreter will search for a machine code function of that name.

If the definition of name cannot be found, then the error 'Not Found' will be reported if name was defined in the past, but the definition line has since been deleted.

If name has never been defined in the current SuperBASIC program, then it will be treated as a normal variable and relevant error messages reported.

Under SMS in both instances the value 0 will be returned (name is treated as an undefined variable).

The method of searching for a FuNction means that if a SuperBASIC FuNction is defined with the same name as a machine code one, the machine code one will no longer be available, and when the SuperBASIC FuNction is removed (for example with NEW), that keyword will no longer have any effect. If entered as a direct command, even the in-line structure will not have any effect unless it is also called on the same line, as the interpreter must jump to the relevant DEFine FuNction statement when the function is called.

If a DEFine FuNction statement appears in a program, if the code is not called, program flow will continue from the statement following the next END DEFine - it is however good practice to keep all definition structures towards the end of a program, and not to place the structure blocks in the middle of program code, as this makes it very difficult to follow the flow of programs.

It is also good programming practice to make FuNctions self-contained and not to jump out of them using GO TOs or GO SUBs (they can of course call other FuNctions and PROCedures).

To call the DEFine FuNction, you merely need to include its name in an expression. If however any parameters are listed in the definition, you will need to pass the same number of parameters in brackets after the name of the FuNction, separated by any valid SuperBASIC separator {ie. comma (,), semicolon (;), backslash (\), exclamation mark (!) or TO }. You can also place a hash (#) before the parameters if you so wish to indicate that it is a channel number.

If not enough parameters are supplied, the program will report 'Error in Expression' when the missing parameter is used, except under SMS where the missing parameters are treated as unset variables and will therefore have the value 0 (if a numeric variable) or else contain an empty string (if a string variable).

If however, too many parameters are passed, the extra parameters are ignored. Parameters are passed by reference which means that the list of items in the DEFine FuNction statement are deemed LOCal to that definition - this means that any previous values of the items are stored whilst the definition block is active. What is more, the type of each item does not actually matter - they assume the type of the passed parameter. For example, the following short program will work without any problems:

```
10 a$=QUERY$('What is your name')
20 DEFine FuNction QUERY$(x)
30   INPUT (x)!b$
40   RETurn b$
50 END DEFine
```

Note though that the name of the FuNction must end with the correct variable type, ie. $ if a string is to be returned, or % if an integer is to be returned (although see note 7 below).

One of the results of passing variables by reference is that if the item is altered within the definition block, if a variable is passed as a parameter, the variable itself will also be altered (although see note 4). This can be shown with the following short program:

```
100 x=10
110 y=Square(x)
120 PRINT x;'^2=';y
130 DEFine FuNction Square(za)
140   za=za*za
150   RETurn za
160 END DEFine
```

This can be avoided by either assigning the item to a temporary variable and then using the temporary variable instead (see the example below), or by passing the variable as an expression, by placing it inside brackets; for example by replacing line 110 with the following:

```
110 y=Square((x))
```

Having passed the necessary parameters to the Function, you can then use each item inside the definition block as normal.

**Example**

A short program to calculate the length of the hypotenuse in a triangle, given the length of its two other sides:

```
100 MODE 4: WINDOW 448,200,32,16: SCALE 100,0,0: PAPER 0 105 CLS: INK 7
110 AT 2,25: UNDER 1: PRINT'Pythagoras calculator': UNDER 0
120 INPUT \\'Enter length of base of triangle:'!base
130 INPUT \\'Enter height of triangle:'!height
140 hypotenuse=Pythag(base,height)
150 INK 4: LINE 50,20 TO 100,20 TO 100,70 TO 50,20
160 INK 7: AT 16,35-LEN(base): PRINT base
170 AT 11,46: PRINT height
180 AT 11,31-LEN(hypotenuse): PRINT hypotenuse
190 :
1000 DEFine FuNction Pythag(x,y)
1010   LOCal x1,y1
1020   x1=x*x:y1=y*y
1030   RETurn SQRT(x1+y1)
1040 END DEFine
```

See what happens if you replace lines 1000 to 1040 with the following:

```
1000 DEFine FuNction Pythag(x,y)
1010   x=x*x:y=y*y
1020   RETurn SQRT(x*y)
1030 END DEFine
```

**NOTE 1**

A FuNction must return a value under all circumstances. If the END DEFine is reached without a value having been returned then SuperBASIC will report an 'error in expression' (-17), specifying the error as having occured at the line containing the END DEFine.

Under SMS the error 'RETurn not in PROCedure or FuNction' will be reported instead.

**NOTE 2**

On pre JS ROMs, you could not define new FuNctions with names which had already been used in the same program.

**NOTE 3**

On pre MG ROMs, any more than nine parameters may upset the program, corrupting it by replacing names with PRINT towards the end of a program. This can however be circumvented by increasing the size of the Name Table by 8 bytes for each name (plus a little more for luck), using the line:

```
CALL PEEK_W(282)+36,N
```

**NOTE 4**

Although a sub-set of a simple string is an expression and therefore will not be altered within a function, a sub-set of a DIMensioned string is not treated as an expression and will therefore be altered!!

**NOTE 5**

Recursive FuNctions (ie. FuNctions which call themselves, or call another PROCedure or FuNction which in turn calls the original FuNction) are allowed (up to 32767 recursions under Minerva). They do however gobble up memory at an amazing rate and can cause problems in compiled SuperBASIC due to the fact that they need an ever-increasing amount of stack space. They should be avoided wherever possible because they are also very slow.

On SMS, if you try to use recursive functions too much, you may end up with the rather esoteric error 'program structures nested too deeply, my brain hurts'! It is however, more likely that you will end up with an 'Out of Error' memory and not be able to do anything else (not even NEW).

**NOTE 6**

The LOCal statement (if used) must appear as the next statement following DEFine FuNction, otherwise an error will be reported. Under SMS if this is not the case, the error 'Misplaced LOCal' will be reported.

**NOTE 7**

SMS and QLiberator do not seem to mind if you do not end the FuNction name with a $ symbol when a string is to be returned and the FuNction will work perfectly well in the compiled version of the program. However, this should be avoided as the program will not work on other QL ROMs and also cannot be compiled with TURBO. For example, take the following program, which works under SMS or when QLiberated.

For other ROMs and TURBO, rename the function to GETSUBDIR$:

```
100 file$='n1_win2_test_bas'
110 test$=GETSUBDIR(file$)
295 :
300 DEFine FuNction GETSUBDIR(s$)
310   IF s$(LEN(s$))<>'_':s$=s$&'_'
320   IF LEN(s$)=5:IF s$(4) INSTR '1234567890':RETurn ''
322   REPeat t_loop
325     root=1
330     FOR x=1 TO LEN(s$)
340       IF s$(x)='_'
350       IF x=3:IF s$(2) INSTR '1234567890':root=3
360       IF x=5:IF s$(4) INSTR '1234567890':root=5
370       IF x>5:IF root=1:s$=PROGD$ & s$:NEXT t_loop
380       IF x=8:IF root=3:root=8
390       END IF
400     NEXT x
410       IF root=1:s$=PROGD$ & s$:NEXT t_loop
415       as$=s$
420       IF root=3:s$=s$(1 TO 3) & PROGD$
425       IF root=3:IF LEN(as$)>3:s$=s$&as$(4 TO):NEXT t_loop:ELSE EXIT t_loop
430     END FOR x
435     EXIT t_loop
440   END REPeat t_loop
445   as$=s$
460   RETurn s$(1 to root)
470 END DEFine
```

**NOTE 8**

Do not try to DEFine one FuNction inside another - although this is actually allowed under most implementations, compilers presume that an END DEFine should be placed before the start of the next DEFine FuNction and it makes programs very difficult to follow.

Under SMS the error 'Defines may not be within other clauses' will be reported when you try to RUN the program.

**NOTE 9**

On Minerva pre v1.96, if you try to link in machine code procedures or functions from inside a DEFine PROCedure or DEFine FuNction block, problems could occur after a CLEAR command.

**WARNING 1**

On most ROMs (at least on JM, MGx, AH and Minerva up to v1.97), a single line recursive FuNction will not respond to the break key. For example:

```
10 DEFine FuNction Root(a): a=2^Root(a)
```

The solution for all ROMs (or all those tested so far!) - insert an additional colon (:) as in:

```
10 DEFine FuNction Root(a)::a=2^Root(a)
```

This is fixed on SMS v2.59+.

**WARNING 2**

All ROMs also suffer from this problem on multiple line recursive FuNctions, where there is no active program line between the definition line and the line which calls the FuNction. For example:

```
10 DEFine FuNction Root(a)
20   a = 2^Root(a)
30 END DEFine
```

The solution here is to insert another active program line at line 15 - for example:

```
15 :
```

or:

```
15 PRINT
```

Do however note that a REMark, DATA or LOCal line at line 15 will not be sufficient as these are not active commands. Again, this is fixed under SMS v2.59.

**WARNING 3**

Except under SMS, if you assign the same name to a FuNction as a resident command, not only will you no longer be able to use the resident command, but it may crash the system!

**SMS NOTES**

In v2.59+, if you fail to create a SuperBASIC function correctly, the error INCOMPLETE DEFine appears (for example if you omit the END DEFine). Prior to v2.89 SMS would only allow a single line DEFine FuNction if END DEFine appeared on the same line. However, although v2.89 would allow a single-line DEFine FuNction without an END DEFine , it would report an error if the END DEFine existed!! Thankfully, v2.90+ fixes this problem, allowing both.

**CROSS-REFERENCE**

*END DEFine* tells the interpreter where the end of the definition block can be found. *RETurn* allows you to return the result of the Function. *DEFine PROCedure* is very similar. *LOCal* allows you to assign temporary variables with the same name as variables used outside the definition block. *PARUSE* and *PARTYP* allow you to examine the type of the parameters which are passed to the definition block.

## 11.23 DEFine PROCedure

| | |
|---|---|
| Syntax | DEFine PROCedure name [(item $^{*}$[,item$^{i}$]$^{*}$ )] |
| Location | QL ROM |

This command marks the beginning of the SuperBASIC structure which is used to surround lines of SuperBASIC code which forms an equivalent to a machine code SuperBASIC procedure, which can be called from within SuperBASIC as a sub-routine. This forms a powerful alternative to GO SUB and helps to make SuperBASIC programs very easy to read and de-bug.

The syntax of the SuperBASIC structure can take two forms:

---

DEFine PROCedure name [(item $^{*}$[,item$^{i}$]$^{*}$ )]: statement $^{*}$[:statement]$^{*}$

or

DEFine PROCedure name [(item *[,item$^i$]* )] *[LOCal var *[,var$^i$]* ]* *[statements]* [RETurn] END DEFine [name]

---

When the specified procedure name is called, the interpreter then searches the SuperBASIC program for the related DEFine PROCedure statement.

If this cannot be found, then the interpreter will look for a machine code procedure of that name.

If the definition of name cannot be found, then the error 'Not Found' will be reported if name was defined in the past, but the definition line has since been deleted.

If name has never been defined in the current SuperBASIC program, then the 'Bad Name' error will be reported. As with FuNctions, the method of searching means that a machine code PROCedure can be overwritten with a SuperBASIC definition and then later lost. Parameters and items are treated in the same manner as with DEFine FuNction. However, please note that calling parameters should not appear in brackets after the name (unless you intend to pass them otherwise than by reference!).

When called, all of the SuperBASIC code within the definition block will be executed until either an END DEFine or RETurn is found, in which case execution will return to the statement after the calling statement. In contrast however, to DEFine FuNction, there is no need for a PROCedure definition block to contain a RETurn statement.

Strictly a PROCedure cannot return a value - however due to the nature of the parameters being passed by reference (see DEFine FuNction), this *is* possible.

**Example**

A simple demonstration program which highlights the fact that a PROCedure or FuNction can actually be recursive (ie. call itself), and also highlights the effect of passing parameters by reference - keep an eye on the values in #0:

```
100 radius=50:height=125:CLS:CLS#0
110 Rndom_circle radius,(height),100
120 AT #0,0,0:PRINT#0,radius,height,100
125 :
130 DEFine PROCedure Rndom_circle(x,y,z)
140   INK RND(7):FILL RND(1)
150   CIRCLE RND (y),RND(z),x
160   FILL 0
170   AT #0,0,0:PRINT#0,x,y,z:PAUSE
180   x=x-RND(5):y=y-1:z=z+1
190   IF x<1:RETurn
200   Rndom_circle (x),y,z
210 END DEFine
```

**NOTE 1**

On pre JS ROMs, you could not define new PROCedures with names which had already been used in the same program.

**NOTE 2**

On pre MG ROMs, any more than nine parameters may upset the program, corrupting it by replacing names with PRINT towards the end of a program. This can however be circumvented by increasing the size of the Name Table by 8 bytes for each name (plus a little more for luck), using the line:

```
CALL PEEK_W(282)+36,N
```

**NOTE 3**

Recursive PROCedures (ie. PROCedures which call themselves, or call another PROCedure or FuNction which in turn calls the original PROCedure) are allowed (up to 32767 recursions on Minerva). They do however gobble up memory at an amazing rate and can cause problems in compiled SuperBASIC due to the fact that they need an ever-increasing amount of stack space. They should be avoided wherever possible. On SMS, if you try to use recursive functions too much, you may end up with the error 'program structures nested too deeply, my brain hurts'! It is however, more likely that you will end up with an 'Out of Memory' error and not be able to do anything else (not even NEW).

**NOTE 4**

The LOCal statement (if used) must appear as the next statement following DEFine PROCedure, otherwise an error will be reported. Under SMS if this is not the case, the error 'Misplaced LOCal' will be reported.

**NOTE 5**

Do not try to DEFine one PROCedure inside another - although this is actually allowed under most implementations, compilers presume that an END DEFine should be placed before the start of the next DEFine PROCedure and it makes programs very difficult to follow. Under SMS the error 'Defines may not be within other clauses' will be reported when you try to RUN the program.

**WARNING 1**

As with DEFine FuNction problems do exist with recursive PROCedures which prevent the Break key from working. These problems are fixed by SMS v2.59+

**WARNING 2**

Except under SMS, if you assign the same name to a PROCedure as a resident command, not only will you no longer be able to use the resident command, but it may crash the system!

**SMS NOTES**

From v2.59, as with DEFine FuNction, SMS insists that all PROCedures have an END DEFine statement, even if they are on a single line. If this does not exist, or there is something else wrong with the syntax, then the error 'Incomplete DEFine is reported. The same problems exist in versions prior to v2.90 as with DEFine FuNction for in-line code.

**CROSS-REFERENCE**

Please see *DEFine FuNction*! Also see *END DEFine*. Look at the example for *SWAP* which provides a more practical use of recursive *PROCedure*s.

## 11.24 DEFINED

| | |
|---|---|
| Syntax | DEFINED (anything) |
| Location | BTool |

SuperBASIC is different from other BASIC dialects in that it does not assign a default value to newly introduced but still unset variables (except on SMS which assigns the value Zero to an unset numeric variable and an empty string to an unset string).

This makes it possible for a program to detect if a variable has been properly initialised - an 'error in expression' (-17) is reported if you try to carry out operations on unset variables.

The function DEFINED takes any parameter, no matter what type it is, provided that it is a constant or a variable. DEFINED returns 0 if the parameter was a variable but unset and 1 for defined variables and constant expressions.

**NOTE**

This function does not work on SMS

**CROSS-REFERENCE**

*CLEAR* makes all variables undefined. *PRINT* writes asterisks if unset variables are required to be printed. *TYPE* returns 1, 2 or 3 for undefined variables. See also *UNSET*.

## 11.25 DEG

| | |
|---|---|
| Syntax | DEG (angle) |
| Location | QL ROM |

This function is used to convert an angle in radians into an angle in degrees (which is the system more readily used by humans). Although this will work for any value of angle, due to the very nature of angles, angle should be in the range 0…2*PI, which will return a value in the range 0…360.

**CROSS-REFERENCE**

See *RAD* and the Mathematics section of the Appendix.

## 11.26 DELETE

| | |
|---|---|
| Syntax | DELETE file or<br>DELETE file *[,file$^{i}$]* (THOR XVI) |
| Location | QL ROM, Toolkit II |

The command DELETE removes the stated file from a medium (it actually only deletes its entry from the directory map, which thus allows these files to be recovered if necessary, with a utility such as the Public Domain RETTUNGE_exe, provided that nothing has been written to the disk since it was deleted).

The filename must include the name of the medium, unless you have Toolkit II installed, which alters the command so that the default data device is recognised (see DATAD$).

The command does not report an error if a file was not found! However, if an invalid device is used and Toolkit II is not present, an error will be reported.

The THOR XVI variant of this command follows the original proposal for this command, allowing you to delete several files at the same time by listing each filename, eg:

```
DELETE flp1_boot,flp1_main_bas
```

This latter syntax is accepted on non-Minerva systems, but only the first file will be deleted. If Toolkit II is present, error -15 (bad parameter) is reported.

**Example**

```
DELETE mdv2_PROG_bak
DELETE PROG_bak
```

**CROSS-REFERENCE**

*WDEL* deletes several files interactively. *WDEL_F*, *WDIR* and *TTEDELETE* are also worth a look.

## 11.27 DEL_DEFB

| | |
|---|---|
| Syntax | DEL_DEFB |
| Location | Toolkit II |

QDOS stores information concerning devices and files (and in relation to files, even their contents) in areas of memory known as 'slave blocks' (memory permitting). These slave blocks can be very useful, since when the computer tries to access the same device (or file) again, the access is much quicker, since the relevent details can be loaded from memory, rather than the device - the computer only need look at the device to make certain that it is the same device (or disk) as was previously used.

There are three problems with the use of these slave blocks:

- The initial device access is slowed down as all of the information is effectively read twice - once into memory and once into the program.
- Some disk drives do not support a means of checking if a disk has been amended on a second computer since the last access - meaning that the old version of the information stored in the slave blocks can be loaded instead
- On some hard-disks, the hard-disk itself may not have been altered (you may need to use a command such as WIN_FLUSH).

The command DEL_DEFB can assist with the second of these problems, by deleting all of the slave blocks from memory. Another problem which can be assisted by DEL_DEFB is 'heap fragmentation'. To keep memory tidy, there is an internal list which says where to find which pieces of information. These lists reserve memory and can lead to the phenomenon known as heap fragmentation. The following example demonstrates this:

```
PRINT FREE_MEM
a=ALCHP(10000)
b=ALCHP(10000)
PRINT FREE_MEM
RECHP a
PRINT FREE_MEM
```

First, we noted how much memory is free and then we reserved 20000 bytes of memory in two steps. So there are now 20000 bytes of free memory less. Now, we release the first 10000 bytes and look again at the free memory: it has not actually increased as much as you would have thought! Actually, the memory isn't lost. FREE_MEM returns the largest piece of free memory in RAM. A further ALCHP(10000) would not reduce FREE_MEM in the above example.

Maybe an illustration would make memory management clearer:

```
free memory          |-------------------------|
ALCHP(10000)         |######|------------------|
ALCHP(10000)         |######|######|-----------|
release first block  |======|######|-----------|
```

Key:

```
-- : free memory (returned by FREE_MEM)
## : reserved memory
== : free memory (used for ramdisks)
```

The above-mentioned internal list allocates a small piece of memory which may reduce the largest piece of free RAM available to certain operations which draw large chunks of memory at a time, causing them to fail (out of memory), even though there would be enough memory had the 'drive definition blocks' not fragmented it. The command DEL_DEFB clears these blocks, thus helping to relieve the heap fragmentation.

**NOTE**

Because DEL_DEFB deletes the slave blocks, future device accesses will be slowed!

**WARNING**

Do not use DEL_DEFB if any channels are open to a file.

**CROSS-REFERENCE**

*RECHP*, *CLCHP*, *RELEASE*, *FREE_MEM*, *FREE*. Dynamic RAM disks use effectively all of the free memory. *FORMAT* lists other ways of causing heap fragmentation.

## 11.28 DESPR

| | |
|---|---|
| Syntax | DESPR (bytes) |
| Location | DESPR |

The function DESPR uses an un-documented system call to try and release a given number of bytes from the resident procedure memory on the QL. It is unknown how the ROM tries to decide which bytes to release.

**WARNING**

The system call used only works properly on Minerva ROMs and can crash some versions of the QL. This function should not be used!!

**CROSS-REFERENCE**

Use *RESPR* to allocate resident procedure memory, and do not try to release it at a later stage. Use *ALCHP* and *RECHP* to allocate areas of memory which may be later released.

## 11.29 DESTD$

| | |
|---|---|
| Syntax | DESTD$ |
| Location | Toolkit II |

This function always contains the current default destination device, which is an unofficial QDOS standard and supported by the Toolkit II variants of COPY, WCOPY, WREN, and SPL.

When Toolkit II is initiated, DESTD$='SER'. The default device means that if no other device is stated for the destination file, this device will be used. The default destination device will also be consulted if a device name is supplied but the given file cannot be found on that device.

For example, assuming that DESTD$='flp2_' and DATAD$='ram1_', if you enter COPY example_txt, then the file ram1_example_txt will be copied to flp2_example_txt. This idea can be extended to use prefixes as sub-directories. Sub-directories are separated by underscores, DESTD$ always ends with an underscore.

**CROSS-REFERENCE**

*DEST_USE* and *SPL_USE* both define the default destination device.

*DUP*, *DDOWN* and *DNEXT* allow you to move around the sub-directory tree. *PROGD$* returns the default program device, *DATAD$* returns the default data device. *DLIST* prints all default devices.

## 11.30 DEST_USE

| | |
|---|---|
| Syntax | DEST_USE name |
| Location | Toolkit II |

This command sets the current default destination device to the named directory device. An underscore will be added to the end of the name if one is not supplied. If you supply name as an empty string, this will turn off the default destination directory.

**Example**

```
DEST_USE win1_Quill
```

**NOTE 1**

DEST_USE will overwrite the default set with SPL_USE.

**NOTE 2**

The default devices cannot exceed 32 characters (plus a final underscore) - any attempt to assign a longer string will result in the error 'Bad Parameter' (error -15).

**CROSS-REFERENCE**

Please see *DESTD$* and *SPL_USE*.

## 11.31 DEMO

| | |
|---|---|
| Syntax | DEMO n |
| Location | Shape Toolkit |

As the name suggests, this is only a demonstration. Try the command DEMO 1 and see what happens. Use only odd parameters if you want the screen to be restored to its previous status when the demonstration finishes.

**CROSS-REFERENCE**

The function *ODD* checks if a number is odd or even.

## 11.32 DET

| | |
|---|---|
| Syntax | DET [array] |
| Location | Math Package |

The function DET returns the determinant of a square matrix, meaning that the array (or the part passed) must have two dimensions of equal size, otherwise DET breaks with error -15 (bad parameter).

The array needs to be a floating point array, any other type (including integer arrays) will also produce error -15.

If no parameter is given, DET will use the array that has been supplied to the previously executed MATINV command as its source. If however, this command has not yet been used, DET without a parameter will stop with the error -7 (not found).

You may ask what a determinant is? Briefly speaking, it represents a square matrix by a single number so that important characteristics of the matrix can be deduced from it, eg. the matrix cannot be inverted if the determinant is zero.

**Example**

We will try to approach the eigenvalues of a matrix and list them all (the so-called "spectrum" of a matrix). Due to approximation errors and the simple algorithm employed, there can be more output values than there should be. This can be improved by increasing estep in line 130, but at the cost of speed.

The range of expected eigenvalues (eval1 to eval2) is adapted to the chosen matrix whose random elements only range between 0 and 1. There is no limit for the positive size n of the matrix, n=0 is allowed but does not make sense because CHARPOLY becomes constant:

```
100 CLEAR: RANDOMISE 10: PRINT "Eigenvalues:"
110 n=2: DIM matrix(n,n), one(n,n)
120 MATRND matrix: MATIDN one
130 :
140 eval1=-1: eval2=1: esteps=200
150 eprec<(eval2-eval1)/estep)
160 c1=CHARPOLY(matrix,eval1): count%=0
170 FOR eval=eval1+eprec TO eval2 STEP eprec
180 c2=CHARPOLY(matrix,eval)
190 IF SGN(c1)<>SGN(c2) THEN PRINT eval
200 c1=c2: count%=count%+1
210 AT#0,0,0: PRINT#0,INT(100*count%/esteps);"%"
220 END FOR eval
230 PRINT "absolute fault:"!eprec
240 :
250 DEFine FuNction CHARPOLY(matrix,lambda)
260   LOCal diff(n,n),i
270   FOR i=1 TO n: one(i,i)=lambda
280   MATSUB diff,matrix,one
```

(continues on next page)

(continued from previous page)

```
290   RETurn DET(diff)
300 END DEFine CHARPOLY
```

In practice, a Newton iteration algorithm (or better) would be used.

**CROSS-REFERENCE**

*MATINV* co-operates closely with *DET*, so that for each of them a matrix parameter can be omitted if the other function has been called before; *MATINV* calls *DET* internally. In the example, we used the *MATRND*, *MATIDN*, *SGN* and *MATSUB* keywords which are all part of the same Toolkit.

## 11.33 DEV_NAME

| | |
|---|---|
| Syntax | device$ = DEV_NAME(address) |
| Location | DJToolkit 1.16 |

This function must be called with a floating point variable name as its parameter. The first time this function is called, address *must* hold the value zero, on all other calls, simply pass address *unchanged* back. The purpose of the function is to return a directory device name to the variable device$, an example is worth a thousand explanations.

```
1000 addr = 0
1010 REPeat loop
1020   PRINT "<" & DEV_NAME(addr) & ">"
1030   IF addr = 0 THEN EXIT loop: END IF
1040 END REPeat loop
```

This small example will scan the entire directory device driver list and return one entry from it each time as well as updating the value in 'addr'. The value in addr is the start of the next device driver linkage block and *must not be changed* except by the function *DEV_NAME*. If you change addr and then call *DEV_NAME* again, the results will be very unpredictable.

The check for addr being zero is done as this is the value returned when the final device name has been extracted, in this case the function returns an empty string for the device. If the test was made before the call to *DEV_NAME*, nothing would be printed as addr is zero on entry to the loop.

Please note, every QL has at least one device in the list, the 'MDV' device and some also have a device with no name as you will see if you run the above example (not the last one as it is always an empty string when addr becomes zero).

The above example will only show directory devices, those that can have DIR used on them, or *FORMAT* etc, such as WIN, RAM, FLP, FDK etc, however, it cannot show the non-directory devices such as SER, PAR (or NUL if you have Lightning), as these are in another list held in the QL.

**Note**

From version 1.14 of DJToolkit onwards, there is a function that counts the number of directory devices present in the QL. See *MAX_DEVS* for details.

**CROSS-REFERENCE**

*MAX_DEVS*.

## 11.34 DEVICE_SPACE

| | |
|---|---|
| Syntax | DEVICE_SPACE ([#]channel) |
| Location | Turbo Toolkit |

This function returns the number of unused bytes on the medium (disk, hard disk or microdrive) to which the specified channel is open. The channel must relate to an open file on a directory device (otherwise junk figures may be returned).

**Example**

A short routine which saves an area of memory to disk, with error checking.

```
100 OPEN #3,'CON_448X200A32X16'
110 CLS #3
120 FILE$='FLP1_MEMORY_BIN'
130 FILE_SIZE=20000: ADDR=ALCHP(FILE_SIZE)
140 REPEAT LOOP
150   INPUT #3,'ENTER FILENAME TO SAVE MEMORY TO : [DEFAULT=';(FILE$);']';F$
160   IF F$='': F$=FILE$: ELSE FILE$=F$
170   OPEN_STATE=DEVICE_STATUS(2,FILE$)
180   IF OPEN_STATE=-20: PRINT #3,'DEVICE IS READ ONLY': NEXT LOOP
190   IF OPEN_STATE=-11: PRINT #3,'DEVICE IS FULL': NEXT LOOP
200   IF OPEN_STATE=-8
210     INPUT #3,'DO YOU WANT TO DELETE EXISTING FILE ? (Y/N)';A$
220     IF A$=='Y'
230       CH=FOP_IN(FILE$)
240     ELSE
250       PRINT #3;'ENTER NEW FILENAME': PAUSE 100
260       NEXT LOOP
270     END IF
275   ELSE
277     CH=FOP_NEW(FILE$)
280   END IF
300   IF CH<0:REPORT #3: NEXT LOOP
305   FREE_SPACE=DEVICE_SPACE(#CH)
307   IF OPEN_STATE=-8: FREE_SPACE=FREE_SPACE+FLEN(#CH)
310   IF FREE_SPACE>=FILE_SIZE: PRINT#3,'SAVING FILE': EXIT LOOP
320   PRINT #3;'NOT ENOUGH ROOM ON DEVICE'
330   CLOSE #CH
335   IF OPEN_STATE<>-8: DELETE FILE$
340 END REPEAT LOOP
350 CLOSE #CH
355 DELETE FILE$
360 SBYTES FILE$,ADDR,FILE_SIZE
```

**NOTE**

Current versions of this fuction have difficulty returning the amount of space on large capacity drives, such as hard disks. It assumes that a sector contains 512 bytes and will only cope with a maximum of 65535 sectors.

**CROSS-REFERENCE**

See *FOPEN* and *DEVICE_STATUS* for more details on accessing directory devices. *DEVTYPE* finds out what type of device a channel is looking at.

## 11.35 DEVICE_STATUS

| | |
|---|---|
| Syntax | DEVICE_STATUS ([open_type,] filename$) |
| Location | Turbo Toolkit |

This function returns a value representing the current status of the device to which the specified filename$ points and can be used to check if an error will be generated when you try to access the given file. The open_type defaults to 2 and can take the following values:

- -1: Use for OPEN or OPEN_NEW
- 0: Use for OPEN
- 1: Use for OPEN_IN
- 2: Use for OPEN_NEW

If an open_type of 2 is specified, then the function will try to create the file and return an error code if this is not possible. The temporary file is deleted in all cases.

If an open_type of 0 is specified then the function will try to open the file for exclusive two way access and report any errors.

If an open_type of 1 is specified the function opens the specified file for read only access, which means that it does not care if a channel is already open to the file from another program.

Finally, if an open_type of -1 is specified, the function will first of all try to open a channel to the file, returning -8 if it already exists and can therefore be read.

If it does not already exist, the function will try to create a temporary file and then read back from it to check that the device can be written to and read from, reporting any errors which are found. Any temporary file is then deleted by the function. This enables IN USE and bad or changed medium errors can be detected!

If the open is successful the amount of free space on the drive is returned akin to DEVICE_SPACE, otherwise a standard QDOS error code is returned.

**NOTE 1**

Current versions of this fuction have difficulty returning the amount of space on large capacity drives, such as hard disks. It assumes that a sector contains 512 bytes and will only cope with a maximum of 65535 sectors.

**NOTE 2**

Due to a bug in the QL's hardware, it is impossible to check if a microdrive is read only. In this instance, you will get a bad or changed medium error code (-16).

**CROSS-REFERENCE**

See *DEVICE_SPACE* for an example.

## 11.36 DEVLIST

| | |
|---|---|
| Syntax | DEVLIST [#channel] |
| Location | TinyToolkit |

This command lists all directory devices recognised by the system to the specified channel. A directory device is one which contains files. The default list channel is #1.

**NOTE**

If device names appear in the listing more than once, this means that more than one device driver is loaded. This normally happens with ramdisks ("RAM").

**CROSS-REFERENCE**

Directory devices may be renamed with *CHANGE* (this will have a corresponding effect on *DEVLIST*), whilst any device can be renamed using QRD (this will have no effect on *DEVLIST*). Compare *DLIST*.

## 11.37 DEVTYPE

| | |
|---|---|
| Syntax | DEVTYPE [(#channel)] |
| Location | SMS |

This function returns a value to indicate the type of device the specified channel (default #0) is connected to. At present, you should only look at the first three bits of the return value, ie:

```
x%=DEVTYPE(#channel)
x%=x% && 3
```

The value returned is:

- 0 - a purely serial device
- 1 - a screen device
- 2 - a file system device (ie. it supports file positioning)

Any other values indicate that there is something wrong with the channel (if the value is >2) otherwise, a negative value means that the channel is not open.

**NOTE**

Prior to v2.71, DEVTYPE would return 'End of File' error if the specified channel was attached to a file and the file pointer was at the end of the file.

**CROSS-REFERENCE**

*OPEN*, *OPEN_IN*, *OPEN_NEW* and *OPEN_OVER* allow you to open channels.

## 11.38 DEV_LIST

| | |
|---|---|
| Syntax | DEV_LIST [#channel] |
| Location | DEV device, GOLD CARD, ST/QL, SMS |

This command lists all DEV_USE definitions to the given channel, default #1. You can also use a public domain utility, DEV Manager, to set and list DEV definitions on a per-program basis.

**Example**

DEV_LIST for example 4a of DEV_USE prints:

```
DEV1_ FLP2_SOURCES_ -> DEV4_
DEV2_ FLP1_COMPILER_ -> DEV3_
DEV3_ FLP1_COMPILER_UTILS_ -> DEV4_
DEV4_ RAM1_ -> DEV5_
DEV5_ FLP1_SOURCES_OTHER_ -> DEV1_
```

**CROSS-REFERENCE**

*DEV_USE*, *DEV_USE$*, *DEV_NEXT* Compare *DEVLIST* and *DLIST*.

## 11.39 DEV_NEXT

| | |
|---|---|
| Syntax | DEV_NEXT (n) n=1..8 |
| Location | DEV device, GOLD CARD, ST/QL, SMS |

The function DEV_NEXT returns the number of the next DEVice where a given DEV will look on next if a file was not found. If a DEV is not defined or has the search option disabled, DEV_NEXT returns zero (0), otherwise an integer from 1 to 8 will be returned.

**Example**

A program which lists a search path:

```
100 INPUT "Which DEV device (1..8)?"!n
110 IF n<1 OR n>8 THEN RUN
120 DIM checked%(8)
130 REPeat SPate
140   IF NOT DEV_NEXT(n) OR checked%(n): EXIT SPate
150   PRINT DEV_USE$(n)
160   checked%(n)=1
170   n=DEV_NEXT(n)
180 END REPeat SPate
```

If you understood this example, then you will know exactly how the DEV device works.

**CROSS-REFERENCE**

*DEV_USE$*, *DEV_LIST*, *DEV_USE*

## 11.40 DEV_USE

| | |
|---|---|
| Syntax | DEV_USE n,drive [,next_dev] n=1..8 or<br>DEV_USE [n](SMS v2.70+ only) or<br>DEV_USE [drivetype] |
| Location | DEV device, GOLD CARD, ST/QL, SMS |

The DEV device is a universal method of driving devices (MDV, FLP, WIN, MOS, ROM), and thus allows old software to recognise default devices/ sub-directories as well as simplifying the use of them. It also introduces fully programmable search paths to QDOS.

There are eight separate DEV drives available, DEV1_ to DEV8_, each of which can point to a real drive and directory defined with DEV_USE.

The first parameter of the command is the number of the DEV device to be defined.

The second specifies what DEVn_ represents.

There is no default and nothing is predefined, but DEV_USE permits only valid drives and directories. Any default devices (DATAD$, PROGD$ etc) are not recognised so the full directory name (including the drive name) must be stated.

There is one special second parameter, the empty string, which removes the definition of the given DEV device; there is no error reported if it was not defined.

The second syntax (SMS v2.70+) also allows you to remove a definition by simply passing the number of the DEV device to delete.

**Example 1**

```
DEV_USE 1,flp1_
DEV_USE 2,flp1_SUBDIR_
DEV_USE 3,flp1_SUBDIR
DEV_USE 4
```

Each time that DEV1_ is accessed, the actual drive which will be accessed is FLP1_, eg. DIR DEV1_ lists a directory of FLP1_.

However, LOAD DEV2_BOOT will load FLP1_SUBDIR_BOOT but especially note that LOAD DEV3_BOOT would try to load FLP1_SUBDIRBOOT (that's not a typing error).

You can therefore see the importance of specifying the underscore! Whereas DATA_USE always adds an underscore to the supplied parameter if there one was not specified, DEV_USE does not. Please pay attention to this difference!

DEV_USE's third parameter is optional and ranges from 0 to 8. This is used to specify another DEV device which should be tried if DEVn_ was accessed for a given file, but the file was not present on that DEV device.

In all other cases: if the drive in general is currently inaccessible (eg. open for direct sector read/write), the file is damaged or already in use, the DEV device will stop with the appropriate error message, and behave as normal in such situations.

**Example 2**

```
DEV_USE 1,flp1_,2
DEV_USE 2,flp1_TEST_
```

VIEW DEV1_Prog_bas will first try to show FLP1_Prog_bas and if it did not find that file, it will then try DEV2_Prog_bas which is actually FLP1_TEST_Prog_bas. If this also fails, VIEW stops with a 'Not Found' error. You might notice that this could lead to an endless search if DEV2_ was told to jump back to DEV1_ if flp1_TEST_Prog_bas also did not exist.

**Example 3**

```
DEV_USE 1,flp1_,2
DEV_USE 2,flp1_TEST_,1
```

Luckily, this is no problem - the DEV device never circles back to a DEV which has already been tried. So, using the definition given for example 3, VIEW DEV1_Prog_bas looks for FLP1_Prog_bas, then FLP1_TEST_Prog_bas and breaks with 'Not Found' because DEV1_ has already been tested.

That's why a DEV device cannot point to another DEV device, DEV_USE 1,DEV2_ is illegal.

It is advisable to give seldom used drives and directories a lower search priority because it naturally takes a little time to scan through a directory for a file. Preferred directories and fast RAM disks (which take next to no time to check for a file) should be checked before the less often-used directories are looked at.

**Example 4a**

```
DEV_USE 1,flp2_SOURCES_,4
DEV_USE 2,flp1_COMPILER_,3
DEV_USE 3,flp1_COMPILER_UTILS_,4
DEV_USE 4,ram1_,5
DEV_USE 5,flp2_SOURCES_OTHER_,1
```

The search path for DEV1_ is:

- FLP2_SOURCES_ go to DEV4_
- RAM1_ go to DEV5_
- FLP2_SOURCES_OTHER_ go to DEV1_, we already tried that, so stop

The search path for DEV2_ is:

- FLP1_COMPILER_go to DEV3_
- FLP1_COMPILER_UTILS_ go to DEV4_
- RAM1_ go to DEV5_FLP2_SOURCES_OTHER_ go to DEV1_
- FLP2_SOURCES_ go to DEV4_, already checked, so stop.

You see that the two search paths for DEV1_ and DEV2_ are connected in one way. This rather complicated example suggests that it would be useful to set the data and program device as follows:

**Example 4b**

```
DATA_USE DEV1_
PROG_USE DEV2_
```

Taking into account that Toolkit II tries the program device after failing to find a file on the data device, a VIEW TEXT will first search through the DEV1_ list and then DEV2_ (thus looking through all DEVs) while EX PROG_exe stops after checking DEV2_ and its connected DEVs.

All operations creating or deleting files will only check for the original DEV definition and ignore the optional paths. This prevents files from being unintentionally deleted or overwritten.

Given the settings of examples 4a and 4b, OPEN_IN #3,DEV1_TEXT will act as VIEW did before whereas OPEN_NEW #3,DEV1_TEXT creates FLP2_SOURCES_TEXT or reports an error/asks if you want to overwrite (if necessary).

DELETE always behaves as an exception in that it does not report an error if a file was not found.

You may have noticed that the third parameter allows a wider range than the DEV number. A zero as the third parameter simply does the same as no third parameter.

The third syntax of DEV_USE is completely different from the first two. It is analogous to the FLP_USE, RAM_USE and NFS_USE commands and allows you to use a different three letter code for the DEV device:

```
DEV_USE fry.
```

DEV1_ is now called fry1_, DEV2_ fry2_ and so on. However, you can also use existing devices.

**Example 4c**

```
DEV_USE FLP
```

Now, things become really complex. With examples 4a and 4b still being valid, FLP1_ actually refers to FLP1_SOURCES_, searching through all the other DEV definitions as well in order to find a file.

The definitions of DEV1_ as FLP1_SOURCES_ and DEVs as FLP do not collide. However, if you issued FLP_USE DEV, FLP1_ and DEV1_ are not known any more until FLP_USE FLP restores the default name for disk drives.

Equally, DEV_USE DEV restores the DEV name (although this can be abbreviated by a DEV_USE without any parameters).

**Example 5**

```
DEV_USE DEV1_
```

refers to the true DEV1_ again, DEV2_, DEV3_, . . . , too.

Renaming DEV has been mainly implemented to convince existing software believing that a directory file always has five letters (eg. MDV1_) to accept sub-directories of level-2 drivers as directory files, too.

**NOTE**

At least up to v2.01, the DEV device does not work fully on any machine. For example WSTAT lists the file names but not the other information: DEV_USE 1,FLP1_TEST_ WSTAT DEV1_

**CROSS-REFERENCE**

*DATA_USE*, *PROG_USE*,*DEV_USE$*, *DEV_NEXT*. *DEV_USEN* is the same as the third syntax on SMSQ/E. *DEV_LIST* lists all DEV definitions. *MAKE_DIR* and the DMEDIUM_XXX commands, starting with *DMEDIUM_DENSITY* are also interesting.

## 11.41 DEV_USEN

| | |
|---|---|
| Syntax | DEV_USEN [drivetype] |
| Location | SMSQ/E |

This command is provided on SMSQ/E to allow you to alter the three letter reference used to access the DEV devices. If no parameter is specified, then the name reverts to DEV.

**Example**

```
DEV_USE 2,'win1_progs_'
DEV_USEN 'flp' DIR flp2_
```

This will provide a directory of win1_progs_ - this can be reset with:

```
DEV_USEN
DIR dev2_
```

**CROSS-REFERENCE**

*DEV_USE* allows you to do the same thing. *FLP_USE* allows you to alter the three letter description for floppy disks.

## 11.42 DEV_USE$

| | |
|---|---|
| Syntax | DEV_USE$ (n) where n=1..8 |
| Location | DEV device, GOLD CARD, ST/QL, SMS |

The DEV_USE$ function returns the actual drive and directory for the number of a DEV device. If a device was not defined, DEV_USE$ will return an empty string “”, LEN(DEV_USE$(n))=0.

**Example**

A listing of all DEV definitions:

```
100 UNDER 1: PRINT "DEV";: UNDER 0
110 PRINT " ";: UNDER 1: PRINT "definition": UNDER 0
120 found=0
130 FOR n=1 TO 8
140 IF LEN(DEV_USE$(n)) THEN
150   PRINT n TO 5;DEV_USE$(n)
160   found=1
170 END IF
180 END FOR n
190 IF NOT found: PRINT "no DEVs defined"
```

**CROSS-REFERENCE**

*DEV_NEXT*, *DEV_LIST*,*DEV_USE*.

## 11.43 DIM

| | |
|---|---|
| Syntax | DIM array (index1 *[index$^i$]* ) *[,array$^j$ (index *[index$^j$]* )]* |
| Location | QL ROM |

The command DIM allows you to set up one or more SuperBASIC arrays which may be of string, integer or floating point type. Each index must be an integer in the range 0…32767.

### 11.43.1 Numeric Arrays

Each index defines the maximum number of elements (less one) in any one direction, which provides the following examples:

```
DIM a(10)
```

sets up a floating-point array a containing 11 elements, a(0) to a(10);

```
DIM z%(10,10)
```

sets up a two dimensional integer array z% containing 121 elements, z%(0,0) to z%(10,10) Each element can hold a different number which can later be accessed by specific reference to each index. When the array is set up, each element is set to zero.

### 11.43.2 String Arrays

String arrays are peculiar and have various differences to both un-dimensioned strings and number arrays.

In a string array, the final index contains the maximum length of a string, rounded up to the next even number (an attempt to assign a longer string to one of the array elements will result in a truncated string). For example:

```
DIM a$(10)
```

sets up a one-dimensional string array a$ with a maximum of 10 characters. This is similar to a$=FILL$(" ",10), except that a$ now has a maximum length;

```
DIM z$(10,9)
```

sets up a two-dimensional string array, which can hold 11 strings (z$(0) to z$(10)), each up to 9 characters long.

When a string array is set up with DIM, each entry is set to a nul string (""). The zero'th element of each string array contains the actual length of that string, for example:

```
DIM a$(10,10): a$(1)='Hello': PRINT a$(1,0)
```

will return the value 5, as will PRINT LEN(a$(1)).

If a$ is undimensioned and a$='Hello World', PRINT a$(0) does not generally work and will result in an 'Out of Range' error, except under SMS v2.60+ and Minerva where PRINT a$(0) is the same as PRINT LEN(a$).

### 11.43.3 Sub-Sets of Arrays

Sub-sets of arrays can also be accessed, for example:

```
PRINT z$(0 TO 2)
```

will print the first three strings stored in the array z$.

### 11.43.4 Omitting Indices

This can be one of the most difficult parts of SuperBasic from the point of view of making programs compatible on all implementations of SuperBASIC and also making programs work the same under the interpreter and when compiled.

The ST/QL Emulators (with E-Init v1.27 or later) follow the same rules as SMS. If an index is omitted, SuperBASIC inserts a default index of:

```
0 TO DIMN (array,index_no)
```

For example, if array is a two-dimensional array, array(1) is the same as using the form array (1,0 TO DIMN(array,2)). Unfortunately, string arrays are slightly different when using the last index.

If the last index is omitted, this defaults to an index of:

```
1 TO LEN(array$(x))
```

However, except on SMS, if a start descriptor is specified, but not an end one, the last index defaults once again to: start_descriptor TO DIMN(array$,index_no). On SMS this defaults to start_descriptor TO LEN(array$(x).

Even more oddly, except on SMS and Minerva, if a start descriptor is omitted, but an end descriptor specified, the index defaults to: 0 TO end_descriptor normally resulting in an error. (On SMS and Minerva this defaults to 1 TO end_descriptor).

However, except on SMS and Minerva, if neither a start nor end descriptor are specified, but the TO itself is specified, this defaults to 0 TO DIMN (array$,index_no), again normally causing an error.

On SMS this defaults to 1 TO LEN (Array$ (x)

On Minerva this defaults to 1 TO DIMN (array$,index_no)

This creates the following result:

```
DIM a$(10):a$='Hello' INK 7:PAPER 0
STRIP 2
```

```
PRINT a$
```

Prints ‘Hello’ => a$ (1 TO LEN(a$) (On all implementations)

```
PRINT a$(1 TO)
```

Prints ‘Hello ‘ => a$(1 TO DIMN(a$,1)) (except on SMS, where it prints ‘Hello’, unless the program is compiled with Qliberator in which case the original system is adopted).

```
PRINT a$(TO)
```

Results in 'Out of Range' => a$(0 TO DIMN(a$,1)) (except on SMS, where it prints 'Hello', and on Minerva where it prints 'Hello ' In both cases, if the program is compiled with Qliberator it still reports an error).

```
PRINT a$( TO 5)
```

Results in 'Out of Range' => a$(0 TO 5) (again on SMS and Minerva it still prints 'Hello', unless the program is compiled with Qliberator, which reports an error).

### 11.43.5 Un-Dimensioned Strings

You can use sub-sets of un-dimensioned strings, for example:

```
a$='Hello World':PRINT a$(1 TO 5)
```

However, such subsets are always treated as expressions, which means that if such a subset was passed as a parameter to a FuNction or PROCedure (see DEFine FuNction), it cannot be passed by reference and the string will remain unaltered by the FuNction/PROCedure.

Compare this with a sub-set of a string array, which will be altered (this sub-set exists as a sub-array). Please see Example 3 below.

The handling of descriptors is also different with un-dimensioned strings. If neither a start nor an end descriptor are specified, this, like string arrays, defaults to:

```
1 TO LEN(string$)
```

However, if the start descriptor is specified, but not the end descriptor, this defaults to:

```
start_descriptor TO LEN(string$)
```

However, if the start descriptor is omitted (whether the end descriptor is specified or just TO is used), unless you have Minerva or SMS, this defaults to:

```
0 TO end_descriptor
```

and:

```
0 TO LEN(string$)
```

respectively, both of which cause an 'out of range' error.

On Minerva and SMS however, this defaults to:

```
1 TO end_descriptor
```

and:

```
1 TO LEN(string$)
```

respectively, thus avoiding this error.

This leads to the following result:

```
CLEAR
x$='Hello'
INK 7: PAPER 0: STRIP 2
```

```
PRINT x$
```

This Prints 'Hello'.

```
PRINT x$(1 TO)
```

This prints 'Hello'

```
PRINT x$(TO)
```

This results in 'Out of Range' or 'Hello' on Minerva and SMS.

```
PRINT x$( TO 10)
```

This results in 'Out of Range' or 'Hello' on Minerva and SMS.

### 11.43.6 ERRORS

Due to the complexity of DIM, we felt that it would be useful to explain some of the various errors which may be reported. SMS has an improved Interpreter which reports more intelligible error codes, therefore those have been used:

**Only arrays may be dimensioned**

This occurs when you try to DIM the name of a procedure or function. It also occurs if you try to use DIM on one of the parameters of a procedure or function and that parameter is not itself a dimensioned variable:

```
   100 DIM x(10)
   110 c=1:test x,1
   130 DEFine PROCedure test (a,b)
   140   DIM b(10)
   150 END DEFine

On other implementations, 'Bad Name' is reported in both instances.
```

**Procedure and function parameters may not be dimensioned**

This only happens as in the example above where you try to DIMension a variable which is in fact one of the parameters from the DEFine PROCedure or DEFine FuNction line (eg. line 140). Here, if you pass a dimensioned variable, eg: TEST 1,x, you get this error under SMS. Also see note 7. On other implementations no error is reported and the problems listed in Note 7 occur.

**SBASIC cannot put up with negative dimensions**

This occurs if you try to use a negative index, for example: DIM x(-10) On other implementations 'Out of Range' is reported.

**Dimensional overflow - you cannot be serious!**

Too many indices have been specified in the DIM statement - refer to Appendix 8.

**Error in Expression**

SMS has either been unable to make any sense of the index, or else it exceeds 32767. On other ROMs you will get the error 'Overflow' if index exceeds 32767.

**Unknown function or array**

This is generally reported of you try to use a Procedure name as the index. Other implementations report 'Error in Expression'

### 11.43.7 EXAMPLES

**Example 1**

A program which acts as a simple quiz program, but shows off some of the best features of using arrays - it is simplicity itself to add new questions and answers to this quiz (just amend quest and target and add the new questions and answers as DATA at the end of the program):

```
100 MODE 8:WINDOW 512,256,0,0:PAPER 0:CLS
110 WINDOW 448,200,32,16
120 quest=5:target=5
130 DIM question$(quest,50),option$(quest,3,25),answer(quest)
140 RESTORE
150 FOR i=0 TO quest-1
160   READ question$(i)
170   FOR j=1 TO 3:READ option$(i,j)
180   READ answer(i)
190 END FOR i
200 REPeat main_loop
210   score=0
220   FOR i=1 TO 7,1:BORDER 10,i:PAUSE 2
230   PAPER 6:CLS:INK 2:AT 3,10:UNDER 1:CSIZE 2,1
240   PRINT 'QUIZ EXAMPLE':CSIZE 2,0:UNDER 0
250   INK 0:AT 0,20:PRINT 'SCORE = ';score
260   DIM asked(quest)
270   REPeat loop
280     opt=RND(1 TO quest)
290     IF asked(opt)=1 THEN
300       FOR j=1 TO quest
310         IF asked(j)=0:opt=j:EXIT j
320         NEXT j
330         DIM asked(quest):NEXT loop
340       END FOR j
350     END IF
360     asked(opt)=1
370     AT 4,0:CLS 2
380     ask_question(opt)
390     reply=get_answer
400     AT 16,0:PAPER 2:INK 7
410     IF reply=answer(opt-1)
420       PRINT 'Correct':score=score+1
430     ELSE
440       PRINT 'Wrong!':score=score-1
```

(continues on next page)

(continued from previous page)

```
450     END IF
460     PAPER 6:INK 0
470     AT 0,20:PRINT 'SCORE = ';score
480     PAUSE
490     IF score=target OR score<0:EXIT loop
500   END REPeat loop
510   PAPER 0:CLS
520   INK 2+2*(score=target):CSIZE 3,1
530   IF score=target
540     PRINT 'Congratulations'
550   ELSE
560     PRINT 'Oh Dear'
570   END IF
580   CSIZE 2,0:INK 7
590   PRINT \\'Try again?? -----> y/n'
600   REPeat keys
610     key$=INKEY$(-1):IF key$ INSTR 'yn':EXIT keys
620   END REPeat keys
630   IF key$=='n':STOP
640 END REPeat main_loop
645 :
650 DEFine PROCedure ask_question(no)
660   LOCal i
670   AT 6,0:start_word=1:end_word=1
680   no=no-1
690   REPeat quest_loop
700     FOR char=start_word TO question$(no,0)
710       IF question$(no,char)=' ':EXIT char
720     END FOR char
730     end_word=char
740     PRINT !question$(no,start_word TO end_word)!
750     IF end_word=question$(no,0):EXIT quest_loop
760     start_word=end_word+1
770   END REPeat quest_loop
780   REPeat opt_loop
790     PRINT \
800     FOR i=1 TO 3
810       PRINT TO 5;i;' = ';option$(no,i)
820     END FOR i
830 END DEFine
835 :
840 DEFine FuNction get_answer
850   REPeat keys
860     key$=INKEY$(-1)
870     IF key$ INSTR '123':RETurn key$
880   END REPeat keys
890 END DEFine
895 :
900 DATA 'The standard Sinclair QL has how much memory?'
```

(continues on next page)

(continued from previous page)

```
910 DATA '16K','128K','640K',2
920 DATA "What was the name of Sinclair's first computer?"
930 DATA 'Z80','ZX81','ZX80',3
940 DATA 'Who is the main person responsible for QDOS?'
950 DATA 'T.Tebby','J.Jones','C.Sinclair',1
960 DATA "Which company created the QL's Gold Card?"
970 DATA 'Miracle Ltd.','Digital Precision Ltd.','Mercury',1
980 DATA 'Who is the main person responsible for SuperBASIC?'
990 DATA 'T.Tebby','J.Jones','C.Sinclair',2
```

Some of you may have noticed that we have used DIM option$(quest,3,25) when we could have used DIM option$(quest,2,25). The reason for this is to make it easier to check the text - try PRINT option$ and you will see that each set of three options is separated by a blank string.

**Example 2**

Take the two arrays set up with:

```
DIM x(2,3,4),x$(2,4,6).
```

The following sub-arrays produce the following equivalents:

```
x(TO, TO 2, 1 TO) => x(0TO 2,0 TO 2,1 TO 4)
x$(1 TO 2, TO 2) => x$(1 TO 2,0 TO 2,1 TO LEN(x$(..)))
x$(TO 2, TO,1 TO) => x$(0 TO 2,0 TO 4,1 TO 6)
```

**Example 3**

A short example of the use of sub-arrays and subsets of undimensioned strings:

```
100 DIM a$(11)
110 a$='Hello World'
120 b$='Great World'
130 swap_array a$(1 TO 5),b$(1 TO 5)
140 PRINT a$,b$
150 :
1000 DEFine PROCedure swap_array (a,b)
1010   c$=b: b=a: a=c$
1020 END DEFine
```

**NOTE 1**

The Turbo compiler alters DIM in compiled programs to enable you to re-dimension arrays without losing their original contents. You may therefore need to physically set the contents of arrays to zero (or nul strings) to ensure that a program works properly when compiled.

**NOTE 2**

On AH ROMs, a floating point array is limited to 384K size.

**NOTE 3**

A variable cannot be used as both a simple variable and an array variable. It is set to an array variable as soon as the line containing the relevant DIM statement is parsed. This means that if a line containing DIM var has been entered, the array var cannot be used until such time as the program has RUN this

line, and in any case, an attempt to use var without array descriptors (eg. var=1) is likely to fail, either resulting in a 'Bad Name' error or 'Error in Expression'.

**NOTE 4**

You cannot assign one array to another. For example:

```
DIM a$ (3,10) , z$(3,10) :z$=a$
```

will report a 'Not Implemented' error.

Compare:

```
z$ ( 1, 1 TO 10 )=a$ (1, 1 TO 10 ).
```

**NOTE 5**

The Turbo and Supercharge compilers insist that strings are all dimensioned as string arrays. They do however also alter the way in which string arrays work so that they operate more like un-dimensioned strings. Un-dimensioned strings may also upset Qliberated tasks!

**NOTE 6**

On pre JS ROMs you cannot use one array as the array sub-script of another in the DIM statement (other than as the first sub-script), for example:

```
DIM a(10):a(3)=10
DIM a$(10,a(3))
```

If you try this, you will find that previous array sub-scripts are set to the value 0, ie. using the above example, a$(0) would be acceptable, whereas a$(2) would cause an error. This will work okay provided that the array is used as the first sub-script, otherwise use a temporary variable. For example:

```
subs=a(3): DIM a$(10,subs)
DIM a$(a(3),10)
```

would both work okay on all ROM versions.

**NOTE 7**

There is a bug in SMS (at least up to v2.88) if you try to DIMension a variable which has been used as a parameter for a PROCedure or FuNction call.

Take the example given above to demonstrate the error 'procedure and function parameters may not be dimensioned'. Now use:

```
CLEAR : TEST a,b
```

no error is reported (although line 140 has no effect).

```
PRINT a,b
```

is equivalent to PRINT a; and any attempt to use b (eg. x=b) reports error in expression, even after CLEAR.

On other ROMs no error is reported. However, the variable passed as a parameter is not re-dimensioned, but some of its elements will no longer be the original value, but very small numbers and any attempt to assign another value to those elements which have been changed may in fact fail!!

**NOTE 8**

Current versions of Qliberator treat all strings in the same way as on the original QL, therefore although a program may RUN fine under the SMS or Minerva intepreter, it may cause problems when compiled. The TURBO and SuperCHARGE compilers treat strings the same as SMS, except see Note 1 and Note 5.

**MINERVA NOTE**

Minerva alters the way in which both dimensioned and undimensioned strings are handled so that:

```
PRINT a$( TO 10)
```

is now acceptable! See above.

Minerva also allows you to slice expressions and numbers. Lines such as:

```
PRINT 'abcd' (2 TO 3)
```

and:

```
a$=101010 (3)
```

will now work. Minerva v1.96+ allows multiple index lists (see SMS Notes).

**SMS NOTES**

SMS alters the way in which both undimensioned and dimensioned strings are handled to make them more sensible (see above). We now await a compiler which handles strings in the same way! SMS says that it no longer handles multiple index lists on assignments (which apparently were allowed on earlier ROM versions - did anyone ever use these?). An example is the line:

```
100 DIM a$(3,4,5)
110 a$(3,4)='Hello'
120 a$(3,4)(2 TO 5)='ELLO'
```

SMS will not let you type in line 120 reporting invalid syntax. To overcome this you have to replace the line with:

```
120 a$(3,4,2 TO 5)='ELLO'
```

In common with Minerva, SMS will now also allow you to slice expressions and numbers. There is a bug in current versions of SMS (at least up to v2.90) when passing string array sub-sets by reference, for example the following program:

```
5 DIM x$(11)
10 x$='Hello World'
15 PRINT x$
20 change x$(1 TO 11)
30 PRINT x$
40 :
1000 DEFine PROCedure change (a$)
1010   a$(1 TO 3)='EXT'
1020 END DEFine
```

At line 30, x$ is shown to be 'HeEXT World'?? It should be 'EXTlo World'. Try making line 20 read:

```
20 change x$
```

Although v2.90 fixes this problem, if you pass a sub-set of an undimensioned string, a worse problem is created. Try deleting line 5 and adding line:

```
1015 PRINT a$: PAUSE
```

before RUNning the program (you may need to use CLEAR beforehand).

**WARNING**

DIM and dimensioned variables can crash the system in certain instances - refer to A8.4 for details of the possible problems and more error messages which can be generated.

**CROSS-REFERENCE**

*DIMN* allows you to find out the maximum sizes of an array. Please see the Appendix on Compatability concerning String Lengths. *LEN* allows you to find the length of a string.

## 11.44 DIMN

| | |
|---|---|
| Syntax | DIMN (array [,dimension] ) or<br>DIMN (array (dimension$^1$ $^*$[,dimension$^i$]$^*$)) |
| Location | QL ROM |

This function allows you to investigate the size of the given index of a specified array.

The first syntax is the most common: it will return the specified dimension (index) used in the original DIM statement when the array was defined. If the index did not exist, then a result of zero is returned.

If dimension is not specified, then the size of the first index is returned.

The second syntax is somewhat obscure and has no practical advantages. This second syntax will not allow you to access the size of the first index. It works by reference to the array itself, for example:

```
PRINT DIMN(a$(1))
```

will return the size of the second index, and:

```
PRINT DIMN(a$(1,1))
```

will return the size of the third index and so forth. Once the number of dimensions used within the DIMN statement has reached the number used by the array, then the value 1 will be returned. If any more are specified, then the error 'Out of Range' will result.

**Examples**

Take an array created with the statement:

```
DIM a$(10,12)
```

The following results will be returned:

```
PRINT DIMN(a$)
```

Will return 10.

```
PRINT DIMN(a$,1)
```

Will return 10.

```
PRINT DIMN(a$,2)
```

Will return 12.

```
PRINT DIMN(a$,3)
```

Will return 0.

```
PRINT DIMN(a$(1))
```

Will return 12.

```
PRINT DIMN(a$(1,1))
```

Will return 1.

```
PRINT DIMN(a$(1,1,1))
```

Will cause an 'Out of Range' error.

**CROSS-REFERENCE**

*LEN* will return the actual length of characters held within a string. *DIM* initialises an array.

## 11.45 DIR

| | |
|---|---|
| Syntax | DIR [#channel,] device or<br>DIR [#channel,] [device] (Toolkit II) or<br>DIR \file [,device] (Toolkit II) |
| Location | QL ROM, Toolkit II |

This command produces a listing to the specified channel (default #1) of all of the files contained on the given device.

The listing gives the name of the device (specified with FORMAT) followed by the number of available sectors/the number of usable sectors; followed by a list of the files in the order they appear on the disk. If you try to get a directory of a ram disk, eg. DIR RAM1_ then the name of the device shown on screen will be RAM1.

If Toolkit II is present, and #channel is a window, a <CTRL><F5> keystroke (pausing output) is generated at the end of each screen full of the listing. You can however also use the third syntax to output the directory to the specified file. If file already exists, you will be given the option of overwriting it. If file doesn't include a device name, the data default directory is used.

The Toolkit II variant also supports the default data directory, which will be used if no device name is given in device, or if the specified device name would result in the error 'Not Found'.

If you have Level-2 or Level-3 device drivers, and there are any sub-directories (created with MAKE_DIR) in the given directory, then if you have Toolkit II present, the names of these sub-directories will appear with the suffix ->.

You can then list the contents of these sub-directories by using DIR with the original device name plus the name of the sub-directory. Level-3 drivers take this one step further in that after the name of the disk in the specified device, appears details of the type of disk being read, ie. MS-DOS or QDOS followed by SD, DD, HD or ED to confirm whether the disk is Single Density, Double Density, High Density or Extra Density. RAM disks are listed as QDOS SD.

**Example 1**

With a cartridge in the left hand microdrive slot,

```
DIR mdv1_
```

might produce the following listing in window #1:

```
QUILL 102/220 sectors
boot
QUILL
install_exe
printer_dat
```

**Example 2**

If Level-2 device drivers are present,

```
DIR flp1_
```

might produce the following:

```
PSION DISK 1000/2880 sectors
QUILL ->
ARCHIVE ->
```

With Level-3 drivers, you would get the same output except the first line would become:

```
PSION DISK QDOS HD
```

```
DIR 'flp1_QUILL'
```

would on both Level-2 and Level-3 drivers, then produce the following output:

```
PSION DISK 1000/2880 sectors
QUILL_boot
QUILL_QUILL
QUILL_install_exe
QUILL_printer_dat
```

**NOTE 1**

With the Toolkit II variant, the <CTRL><F5> will be generated even where the channel is a window which has been opened over the network (eg. n1_scr_200x200), which can cause problems as the slave machine will wait for a key to be pressed!

This can be avoided if you have the command FIXPF (provided as part of the QPTR documentation), which will enable you to re-install the ROM variant of DIR.

Alternatively write the directory to a file and copy the file to the host machine, eg.

```
DIR \ram1_tmp, flp1_
SPL ram1_tmp TO n1_scr_200x200
```

It is even more sophisticated to use a named pipe instead of the temporary file ram1_tmp for the same job:

```
SPL pipe_dir TO n1_scr_200x200
DIR \pipe_dir_1000, flp1_
```

**NOTE 2**

The THOR XVI retains the original QL ROM variant of this command, which does not support the default device, nor does it show sub-directory names.

**NOTE 3**

Unless you have Toolkit II present, the Break key will not have any effect on DIR. Press Break when the listing pauses at the end of a page under Toolkit II (Minerva v1.78+ is supposed to recognise the Break key, but it does not appear to work). The Break key is however recognised in Minerva v1.97 (at least!).

**NOTE 4**

Prior to Toolkit II v2.25, DIR of a Level-2 device driver could cause problems when used inside a TURBO compiled program.

**NOTE 5**

If a directory contains a file with a null string as a name (eg. SAVE flp1_), this file will not be listed on the directory listing. This was used as a form of copy protection on some early QL software, but stops the program from working on a QL with Level-2 or Level-3 Device Drivers as they use this file to store the main directory!

**NOTE 6**

On some versions of Toolkit II, the third variant could cause problems if you supply the name of an existing file to store the directory in, for example:

```
DIR \ram1_XDIR
```

if you said 'N' when asked if it was OK to Overwrite the existing file - the display would be sent to #0 and #0 would then be CLOSEd!! v2.49 of Toolkit II (and possibly earlier) does not cause any problems but does not report an error. v2.85 of SMSQ/E (and possibly earlier) also has no problems but reports the error 'Already Exists'.

**NOTE 7**

Some people try to divide up DIRectory listings by creating files such as:

```
SAVE 'flp1_----------------'
```

However, DIR will only list the files in the order in which they were created if you are using a virgin disk which has not had other files deleted from it already.

**CROSS-REFERENCE**

*DATA_USE* sets the current data default directory, *MAKE_DIR* creates sub-directories, *WDIR* allows wildcard names.

## 11.46 DISCARD

| | |
|---|---|
| Syntax | DISCARD [adr] |
| Location | Memory Toolkit (DIY Toolkit Vol H) |

This command removes memory which has been allocated with RESERVE fairly safely, ensuring that the memory had been allocated with RESERVE and has not already been DISCARDed. If the adr does not point to memory set aside with RESERVE the error 'not found' is returned.

**CROSS-REFERENCE**

See *RESERVE* and *LINKUP*. Also see *CLCHP*, *RECHP* and *DESPR*.

## 11.47 DISP_BLANK

| | |
|---|---|
| Syntax | DISP_BLANK [xblank][,yblank] |
| Location | QVME (Level E-19 Drivers onwards), SMSQ/E for Atari ST & TT (QVME cards only) |

The Atari range of computers can be attached to a wide range of monitors, some of which are able to display higher resolutions than others. A 17" multi-sync monitor, for example, can display resolutions of up to 1024x1024 (depending on make).

The QVME card is unable to detect the various parameters related to monitors and therefore allows you to set your own parameters either from SuperBASIC or by configuring SMSQ/E.

This command is used for setting the margins between the currently displayed QL screen and the edges of the monitor. This difference is known as the overscan (pixels available on the monitor which are currently unused). xblank sets the number of horizontal pixels x2 from the edge of the monitor to the left hand side of the QL screen.

The standard value for a 512x256 screen is 128 pixels (a standard QL monitor linked to an Atari can display a screen 640x480) (640-512)/2=64 pixels from the left hand side of the monitor.

If xblank is omitted or 0, then the original value is left unaltered.

Yblank sets the number of lines x 0.5 from the top of the monitor to the top edge of the QL screen. The standard value is 56, which gives a top margin of (480-256)/2=112 pixels from the top of the screen. If yblank is omitted or 0, then the original value is left unaltered.

**NOTE 1**

If you use DISP_SIZE to alter the size of the displayed QL screen, it will automatically adjust the parameters for the overscan.

**NOTE 2**

If the y parameter is used to alter the number of blank lines, this will override any setting of the line scan rate with DISP_RATE.

**CROSS-REFERENCE**

*DISP_SIZE* allows you to pass these parameters at the same time as amending the size of the displayed QL screen. *DISP_RATE* sets the frame and line scan rates for the display - if this command is used to adjust the line scan rate, this will alter the totoal number of lines.

Both SMSQ/E and QVME include programs to allow you to try out the various settings for the various DISP_… commands.

## 11.48 DISP_INVERSE

| | |
|---|---|
| Syntax | DISP_INVERSE status |
| Location | SMSQ/E for Atari ST & TT |

The Atari range of computers support a high resolution (640x400) monochrome display mode which can be supported under SMSQ/E and SMS2. If SMSQ/E or SMS2 is running on an Atari ST connected to a monochrome monitor (or running on an Atari TT connected to such a monitor, without QVME), then it will automatically start up by loading the monochrome display driver (if available) and set the QL into the monochrome 640x400 display mode. The QL screen can then appear either as white ink on a black background or black ink on a white background. DISP_INVERSE allows you to invert the QL display, with status=0 giving the default white on black and status=1 the black on white display.

**NOTE**

This command is not available on SMS2.

**CROSS-REFERENCE**

*DISP_TYPE* allows you to find out if the monochrome display driver is running. *INVERSE* allows you to print in inverse colours.

## 11.49 DISP_MODE

| | |
|---|---|
| Syntax | DISP_MODE mode |
| Location | SMSQ/E for Q68 |

This function sets the Q68 screen mode where mode can take the following values:

- 0 QL 8 colour mode

  the standard 512 x 256 pixels mode in 8 colours. In this mode you can also set mode 4, with the usual MODE keyword. This is then equivalent to setting DISP_MODE 1.

- 1 QL 4 colour mode

  the standard 512 x 256 pixels in 4 colours mode. In this mode you can also set mode 8, with the usual MODE keyword. This is then equivalent to setting DISP_MODE 0.

- 2 Small 16 bit mode

  512 x 256 pixels in mode 33 (16 bits per pixel).

- 3 Large 16 bit mode

  1024 x 512 pixels in mode 33 (16 bits per pixel). Please note that this mode will slow down the Q68, you should not use this mode when doing something time-critical.

- 4 Large QL Mode 4

  1024 x 768 pixels in QL 4 colours mode (there is no mode 8 in this display mode).

- 5 Aurora compatible 8 bit colours

  1024 x 768 pixels in Aurora 256 colours mode. This allows you to have a big screen with nicer colours while still being reasonably fast (but slower than the QL modes).

- 6 Medium 16 bit mode

  512 x 384 pixels in mode 33 (16 bits per pixel).

- 7 Very large 16 bit mode

  1024 x 768 pixels in mode 33 (16 bits per pixel). Please note that this mode will severely slow down the Q68, you should not use this mode when doing something time-critical.

**CROSS-REFERENCE**

*MODE*

## 11.50 DISP_RATE

| | |
|---|---|
| Syntax | DISP_RATE [frame_scan][,line_scan] |
| Location | QVME (Level E-19 Drivers onwards), SMSQ/E for Atari ST & TT (QVME cards only) |

Due to the multitude of monitors which are available for the Atari ST range, it is necessary to be able to alter the horizontal and vertical scan rates (default = 50Hz, the setting on standard QL monitors).

The first parameter specifies the frame rate (the horizontal scan rate), a setting of 70 (or more) will reduce flicker on most Atari monitors. If omitted or 0, the original value is unchanged.

The second parameter specifies the line rate (the vertical scan rate), although this is normally not required as it is equal to the frame rate multiplied by the total number of lines. If this parameter is omitted or zero, the original is recalculated by reference to the number of lines and the frame rate.

The total number of lines and line rate can be calculated by reference to the following program:

```
100 INPUT #0,'Enter y size of QL screen (DISP_SIZE) ';QLy
110 INPUT #0,'Enter horizontal frame rate (DISP_RATE) ';Frate
120 INPUT #0,'Enter vertical blank pixels setting (DISP_BLANK) ';Blanky
130 Total_y=QLy+Blanky
140 total_lines=Total_y*(Qly/QLy)
150 PRINT 'The total number of displayed lines will be ';total_lines
160 PRINT 'Line scan rate will be ';total_lines*Frate
```

If you use DISP_RATE to set the line scan rate, then using the total number of lines (and hence the blank lines) are recalculated, using the following routine:

```
100 INPUT #0,'Enter y size of QL screen (DISP_SIZE) ';QLy
110 INPUT #0,'Enter horizontal frame rate (DISP_RATE) ';Frate
120 INPUT #0,'Enter vertical line scan rate (DISP_RATE) ';Lrate
130 Total_y=INT(Lrate/Frate)
140 PRINT 'Blank Lines for DISP_BLANK will be ';Total_y-QLy
```

**CROSS-REFERENCE**

*DISP_SIZE* allows you to pass these parameters at the same time as amending the size of the displayed QL screen. *DISP_BLANK* sets the number of horizontal and vertical blank pixels on the edge of the display.

Both SMSQ/E and QVME include programs to allow you to try out the various settings for the DISP_... commands.

## 11.51 DISP_SIZE

| | |
|---|---|
| Syntax | DISP_SIZE |
| Location | QVME (Level E-19 Drivers onwards), SMSQ/E |

This command lets you alter the size of the QL screen being displayed.

The first two parameters allow you to specify the display width in pixels and the height in lines (the normal QL display is DISP_SIZE 512,256). The remaining four parameters are those which can be set using the DISP_RATE and DISP_BLANK commands respectively. The effect of the first two parameters depends upon the system it is being used on:

**Extended Mode4 Emulator**

Any width up to 512 will select the standard QL resolution. Any width over 512 will select the extended resolution (768x280).

**QVME, QXL and QPC**

The width and height of the display can only be altered in increments of 32 pixels and 8 lines respectively. If width is not a multiple of 32 or height is not a multiple of 8, they are made into the nearest feasible size. The minimum size is 512x256 pixels.

**Q68**

Not implemented

**NOTE 1**

If you try to use DISP_SIZE to specify both the line rate and the number of blank lines, the line rate is ignored and calculated according to the internal formula (see DISP_RATE).

**NOTE 2**

DISP_SIZE will not work if you have already used the A_OLDSCR command.

**NOTE 3**

Some combinations of Super Gold Card and AURORA may cause the internal QL clock to run too quickly unless you follow DISP_SIZE by PROT_DATE 0.

**NOTE 4**

This command has no effect if your implementation of the QL does not support higher resolution displays.

**NOTE 5**

Using higher resolution displays will affect the location of the start of the screen (see SCR_BASE) - using DISP_SIZE 512,256 to set the display size back to the normal QL resolution will not set the base of the screen back to 131072 (the normal screen base on a standard QL). See A_OLDSCR.

**NOTE 6**

Be careful when reducing screen resolution size - windows are not resized and therefore you may not be able to see all of a program's windows, or the SuperBASIC cursor!!

**CROSS-REFERENCE**

All of these parameters can be configured in SMSQ/E so that they are available immediately on start-up. See *DISP_RATE* and *DISP_BLANK*.

## 11.52 DISP_TYPE

| | |
|---|---|
| Syntax | DISP_TYPE |
| Location | SMSQ/E |

This function returns a number which allows you to find out the type of display driver which is currently being used. The values returned are:

- 0 Original ST QL Emulator, QL Hardware (either of these two may support MODE 8) plus QXL and QPC. All of these (except the original ST QL emulator) may support higher resolutions.
- 1 Extended Mode 4 Emulator (either 512x256 or 768x280 pixel screen)
- 2 QVME Mode 4 Emulator
- 4 Monochrome display (only two colours)

**CROSS-REFERENCE**

See *DISP_INVERSE*. *MACHINE* and *PROCESSOR* allow you to find out more about the hardware which a program is being run on.

## 11.53 DISP_UPDATE

| | |
|---|---|
| Syntax | DISP_UDPATE x,y |
| Location | QXL (SMSQ only) |

This is an undocumented command and it is uncertain what its parameters do - it affects the rate at which the screen is updated on the QXL. The higher x and y, the faster that the screen is updated (and hence the smoother the graphics), although this also slows down the other parts of the QXL. If x and y are equal to 0, the screen is only updated when you press a key - this allows the QXL to perform complex maths routines very quickly (so long as they do not access the screen).

**NOTE 1**

Prior to SMSQ/E v2.65 if you used DISP_UPDATE with a parameter larger than 1 in MODE 8, this could cause problems on screen.

**NOTE 2**

Using parameters smaller than 0 could lock up some versions of QXL. SCR_PRIORITY is similar under Amiga QDOS.

## 11.54 DISPLAY_WIDTH

| | |
|---|---|
| Syntax | bytes_in_a_line = DISPLAY_WIDTH |
| Location | DJToolkit 1.16 |

This function can be used to determine how many bytes of the QL's memory are used to hold the data in one line of pixels on the screen. Note that the value returned has nothing to do with any *window* width, it always refers to the total *screen* display width.

Why include this function I hear you think? If you run an ordinary QL, then the result will probably always be 128 as this is how many bytes are used to hold a line of pixels, however, many people use Atari ST/QLs, QXL etc and these have a number of other screen modes for which 128 bytes is not enough.

This function will return the exact number of bytes required to step from one line of pixels to the next. Never assume that QDOS programs will only ever be run on a QL. What will happen when new Graphics hardware or emulators arrive? This function will still work, assuming that the unit uses standard QDOS channel definition blocks etc.

For the technically minded, the word at offset $64 in the SCR_ or CON_ channel's definition block is returned. This is called SD_LINEL in 'Tebby Speak' and is mentioned in Jochen Merz's *QDOS Reference Manual* and the *QL Technical Manual* by Tony Tebby et al. Andrew Pennel's book, the *QDOS Companion* gets it wrong on page 61, guess which one I used first!

## 11.55 DIV

| | |
|---|---|
| Syntax | x DIV y |
| Location | QL ROM |

This operator returns the integer part of x divided by y.

If x or y is not an integer, then the given value is rounded to the nearest integer (compare INT).

On non-SMS implementations the answer and both parameters must lie within the range -32768…32767.

On SMS, the answer and both parameters can lie anywhere within roughly -2e9…2e9 (32-bit numbers).

The result of the operation is always rounded down to the next integer ie. x DIV y=INT(x/y). Although this leads to some unexpected results with negative numbers this is so that the formula: x=y*(x DIV y)+(x MOD y) is always true.

If you wish to use 32-bit numbers on non SMS systems, you will need to use the formula: PRINT INT(x/y) instead of PRINT x DIV y if either x or y is outside of the specified range.

**Examples**

```
PRINT 13 DIV 5
```

gives the result 2 (13 divided by 5 is 2.6)

```
PRINT 13.4 DIV 1.5
```

gives the result 6 (13 DIV 2).

```
PRINT -13 DIV 5
```

gives the result -3

**NOTE**

DIV has problems with the value -32768: PRINT -32768 DIV -1 gives the result -32768 on most implementations. On Minerva v1.76 (or later) it gives the correct result, being an overflow error (the answer is +32768 which cannot be stored as a short integer variable). On SMS v2.75+, it returns the value +32768 as DIV can handle the larger numbers!!

**CROSS-REFERENCE**

*MOD* returns the modulus of x divided by y. Also please see the alternative version of *DIV*.

---

## 11.56 DIV

| | |
|---|---|
| Syntax | DIV (x,y) |
| Location | Math Package |

This function returns x/y as an integer in the same way as the ROM based DIV operator. However, this version is not limited to 16-bit integers (-32768..32767). It will happily handle 32-bit integer numbers (-INTMAX..INTMAX, roughly -1E9..1E9). Division by zero is not defined and will produce an overflow message.

**Example**

```
PRINT -40000 DIV 3
```

will produce an error on a standard QL ROM. Instead, you can now use:

```
PRINT DIV(-40000,3)
```

which gives the correct result.

**NOTE 1**

Both variants of DIV can be used in the same program, although the Turbo and Supercharge compilers will not accept this version.

**NOTE 2**

If you try to use a program compiled under Turbo or Supercharge after loading the Math Package, if the program uses the normal SuperBASIC operator MOD or DIV, an error will be generated and the program will refuse to work!

**CROSS-REFERENCE**

*MOD* (as an operator or a function) returns the remainder of a division. Compare the other version of *DIV*.

## 11.57 DJ_OPEN

| | |
|---|---|
| Syntax | channel = DJ_OPEN('filename') |
| Location | DJToolkit 1.16 |

Open an existing file for exclusive use. See *DJ_OPEN_DIR* below for details and examples.

**CROSS-REFERENCE**

*DJ_OPEN_IN*, *DJ_OPEN_NEW*, *DJ_OPEN_OVER*, and *DJ_OPEN_DIR*.

## 11.58 DJ_OPEN_IN

| | |
|---|---|
| Syntax | channel = DJ_OPEN_IN('filename') |
| Location | DJToolkit 1.16 |

Open an existing file for shared use. The same file can be opened by other applications running at the same time. Provided they have a compatible non-exclusive OPEN mode. See *DJ_OPEN_DIR* below for details and examples.

**CROSS-REFERENCE**

*DJ_OPEN*, *DJ_OPEN_NEW*, *DJ_OPEN_OVER*, and *DJ_OPEN_DIR*.

## 11.59 DJ_OPEN_NEW

| | |
|---|---|
| Syntax | channel = DJ_OPEN_NEW('filename') |
| Location | DJToolkit 1.16 |

Create a new file for exclusive use. See *DJ_OPEN_DIR* below for details and examples.

**CROSS-REFERENCE**

*DJ_OPEN*, *DJ_OPEN_IN*, *DJ_OPEN_OVER*, and *DJ_OPEN_DIR*.

## 11.60 DJ_OPEN_OVER

| | |
|---|---|
| Syntax | channel = DJ_OPEN_OVER('filename') |
| Location | DJToolkit 1.16 |

Open existing file but overwrite all the contents. See *DJ_OPEN_DIR* below for details and examples.

**CROSS-REFERENCE**

*DJ_OPEN*, *DJ_OPEN_IN*, *DJ_OPEN_NEW*, and *DJ_OPEN_DIR*.

## 11.61 DJ_OPEN_DIR

| | |
|---|---|
| Syntax | channel = DJ_OPEN_DIR('filename') |
| Location | DJToolkit 1.16 |

All of these DJ_OPEN functions return the SuperBasic channel number if the channel was opened without any problems, or, a negative error code otherwise. You can use this to check whether the open was successful or not.

The filename must be supplied as a variable name, file$ for example, or in quotes, 'flp1_fred_dat'.

They all work in a similar manner to the normmal SuperBasic OPEN procedures, but, DJ_OPEN_DIR offers a new function not normally found on a standard QL.

**DJToolkit Author's Note**

I am grateful to Simon N. Goodwin for his timely article in *QL WORLD volume 2, issue 8* (marked Vol 2, issue 7!!!). I had been toying with these routines for a while and was aware of the undocumented QDOS routines to extend the SuperBasic channel table. I was, however, not able to get my routines to work properly. Simon's article was a great help and these functions are based on that article. Thanks Simon.

**EXAMPLE**

The OPEN routines work as follows:

```
1000 REMark open our file for input
1010 :
1020 chan = DJ_OPEN_IN('filename')
1030 IF chan < 0
1040    PRINT 'OOPS, failed to open "filename", error ' & chan
1050    STOP
1060 END IF
1070 :
1080 REM process data in file here ....
```

DJ_OPEN_DIR is a new function to those in the normal QL range, and it works as follows:

```
1000 REMark read a directory
1010 :
1020 INPUT 'Which device ';dev$
1030 chan = DJ_OPEN_DIR(dev$)
1040 IF chan < 0
1050    PRINT 'Cannot open ' & dev$ & ', error ' & chan
1060    STOP
1070 END IF
1080 :
1090 CLS
1100 REPeat dir_loop
1110   IF EOF(#chan) THEN EXIT dir_loop
1120   a$ = FETCH_BYTES(#chan, 64)
1130   size = CODE(a$(16)):       REMark Size of file name
1140   PRINT a$(17 TO 16 + size): REMark file name
1150 END REPeat dir_loop
```

(continues on next page)

(continued from previous page)

```
1160 :
1170 CLOSE #chan
1180 STOP
```

In this example, no checks are done to ensure that the device actually exists, etc. You could use *DEV_NAME* to check if it is a legal device. The data being read from a device directory file must always be read in 64 byte chunks as per this example.

Each chunk is a single directory entry which holds a copy of the file header for the appropriate file. Note, that the first 4 bytes of a file header hold the actual length of the file but when read from the directory as above, the value if 64 bytes too high as it includes the length of the file header as part of the length of a file.

The above routine will also print blank lines if a file has been deleted from the directory at some point. Deleted files have a name length of zero.

Note that if you type in a filename instead of a device name, the function will cope. For example, you type in 'flp1_fred' instead of 'flp1_'. You will get a list of the files on 'flp1_' if 'fred' is a file, or even, if 'fred' is not on 'flp1_'. If, however, you have the LEVEL 2 drivers (see *LEVEL2* below), and 'fred' is a sub-directory then you will get a listing of the sub-directory as requested.

**CROSS-REFERENCE**

*DJ_OPEN*, *DJ_OPEN_IN*, *DJ_OPEN_NEW*, and *DJ_OPEN_OVER*.

## 11.62 DJTK_VER$

| | |
|---|---|
| Syntax | v$ = DJTK_VER$ |
| Location | DJToolkit 1.16 |

This simply sets v$ to be the 4 character string 'n.nn' where this gives the version number of the current toolkit. If you have problems, always quote this number when requesting help.

**EXAMPLE**

```
PRINT DJTK_VER$
```

## 11.63 DLINE

| | |
|---|---|
| Syntax | DLINE [#ch,] [range *[,range$^{i}$]*](Not SMS)<br>DLINE [range *[,range$^{i}$]*](SMS Only) |
| Location | QL ROM |

This command deletes a given range of lines from the current SuperBASIC program. The range of lines is as per the LIST command. If an empty range (for example DLINE) is specified, no action is taken. When the lines have been deleted, except under SMS, the current listed lines are re-shown in the given channel (default #2), although we cannot see any reason why you would wish this to happen on another channel! On SMS this command has no effect on the display.

**NOTE 1**

DLINE TO is expanded to DLINE 1 TO 32767.

**NOTE 2**

Only Minerva v1.96+ rejects invalid channel parameters.

**NOTE 3**

On Minerva pre v1.98, DLINE to the last line could crash the QL!

**CROSS-REFERENCE**

*EDIT* and *AUTO* allow you to enter lines. *LIST* allows you to view program lines.

## 11.64 DLIST

| | |
|---|---|
| Syntax | DLIST [#channel] |
| Location | Toolkit II |

This command lists all three current default directories (otherwise returned by the DATAD$, PROGD$ and DESTD$ functions) to the specified channel (default #1).

**Example**

```
DLIST
```

possible Output:

```
flp1_Quill_letters_
ram1_
par
```

**NOTE**

Some Toolkit II manuals mention a second syntax: DLIST \file but it seems as though this was never implemented. This should not be a problem since programs can read the same information from the DATAD$, PROGD$ and DESTD$ functions.

**CROSS-REFERENCE**

*DATAD$* (*DATA_USE*), *PROGD$* (*PROG_USE*), *DESTD$* (*SPL_USE* or *DEST_USE*), *DDOWN*, *DUP* Compare *DEVLIST* and *DEV_LIST*.

## 11.65 DMEDIUM_DENSITY

| | |
|---|---|
| Syntax | DMEDIUM_DENSITY [(#channel)] or<br>DMEDIUM_DENSITY (\file) |
| Location | SMSQ/E v2.73+ |

This function returns a number representing the density of the medium on which the specified file or directory is located, or to which the specified channel is open. If no parameter is specified, it looks to channel #3 (or #1 if #3 is not open). An error will occur if the specified channel is not open or the given file does not exist.

The value returned is:

- 0 Non-directory device
- 1 Double Density
- 2 High Density
- 3 Extra Density
- 255 Hard disk or ram disk as they have no density.

**Example**

```
PRINT DMEDIUM_DENSITY(\flp1_)
```

**CROSS-REFERENCE**

*DMEDIUM_NAME$* gives the name of the disk attached to the specified channel. *DMEDIUM_RDONLY* and *DMEDIUM_FORMAT* are also useful.

## 11.66 DMEDIUM_DRIVE$

| | |
|---|---|
| Syntax | DMEDIUM_DRIVE$ [(#channel)] or<br>DMEDIUM_DRIVE$ (\file) |
| Location | SMSQ/E v2.73+ |

This function returns the three letter code representing the device connected to the specified channel or file. If no parameter is specified then it tries #1, unless channel #3 is open in which case it will access #3. If an error occurs, for example you specify a channel which is not open or a file which does not exist, then an error will occur. Luckily due to the fact that directories are stored in files under Level-2 and Level-3 drivers, this means that you can use:

```
PRINT DMEDIUM_DRIVE$(\flp2_)
```

if you wish. If the specified channel is not open to a directory device then an empty string will be returned.

**NOTE 1**

This function does not appear to work 100%, for example on Falkenberg hard disk interfaces it returns 'WINq' - however you can get around this by copying the returned string to another variable and only looking at the first three letters, for example:

```
DRV$=DMEDIUM_DRIVE$
IF DRV$<>"":PRINT DRV$( TO 3)
```

**NOTE 2**

This function will ignore the dev_ device, returning the three letter name of the device to which dev points, for example:

```
DEV_USE 1,'flp1_quill_'
drv$=DMEDIUM_DRIVE$(\DEV1_)
IF drv$<>'':PRINT drv$(to 3)
```

Compare:

```
PRINT DMEDIUM_DRIVE$(\DEV1_)
```

**CROSS-REFERENCE**

*DMEDIUM_NAME$* allows you to find out the name of the disk in the specified drive.

## 11.67 DMEDIUM_FORMAT

| | |
|---|---|
| Syntax | DMEDIUM_FORMAT [(#channel)] or<br>DMEDIUM_FORMAT (\file) |
| Location | SMSQ/E v2.73+ |

This function returns a number representing the operating system under which the medium (or hard disk partition) on which the specified file or directory is located (or to which the specified channel is open) was created. If no parameter is specified, it looks to channel #3 (or #1 if #3 is not open).

The values returned currently are:

- 1 QDOS or SMSQ or SMSQ/E
- 2 DOS or TOS

**NOTE**

This function does not appear to work on Falkenberg hard disk interfaces where it always returns 255.

**CROSS-REFERENCE**

*DIR* will provide this information also on Level-3 device drivers. *DMEDIUM_DENSITY* allows you to check the medium's density. There is currently no way to format a disk in a format other than QDOS or SMSQ/E without the ATARI_rext commands which were available with the ST/QL emulators from Jochen Merz, or without specialist software (some of which is public domain).

## 11.68 DMEDIUM_FREE

| | |
|---|---|
| Syntax | DMEDIUM_FREE [(#channel)] or<br>DMEDIUM_FREE (\file) |
| Location | SMSQ/E v2.73+ |

This function returns the number of free sectors available on the medium on which the specified file or directory is located, or to which the specified channel is open. If no parameter is specified, it looks to channel #3 (or #1 if #3 is not open).

**CROSS-REFERENCE**

*DMEDIUM_TOTAL* allows you to find out the total number of sectors available on the related medium. *DIR* can also be used to obtain this information.

## 11.69 DMEDIUM_NAME$

| | |
|---|---|
| Syntax | DMEDIUM_NAME$ [(#channel)] or<br>DMEDIUM_NAME$ (\file) |
| Location | SMSQ/E v2.73+ |

This function returns the name which was given to the medium on which the specified file or directory is located (or to which the specified channel is open), when that medium was FORMATted. If no parameter is specified, it looks to channel #3 (or #1 if #3 is not open).

**Example**

A routine to re-format a floppy disk with the same details as previously allocated to that disk (except for the files). The drive to format (eg. flp1_) can be passed with or without quotes, due to the use of line 120:

```
100 DEFine PROCedure RE_FORMAT(drv)
110   v$=VER$:IF v$<>'HBA':PRINT #0,'NOT SUPPORTED':PAUSE:RETurn
120   drv$=PARSTR$(drv,1)
130   CH=FOPEN(drv$)
140   IF CH<0:PRINT #0,'File Error - cannot access drive':PAUSE:RETurn
150   IF DMEDIUM_RDONLY(#CH)
160     PRINT #0,'Disk Write Protected, cannot proceed':PAUSE
170     CLOSE #CH:RETurn
180   END IF
190   dname$=DMEDIUM_NAME$(#CH)
200   drv_density=DMEDIUM_DENSITY(#CH)
210   IF DMEDIUM_FORMAT(#CH)<>1
220     PRINT #0,'Not QDOS / SMSQE disk, cannot proceed':PAUSE
230     CLOSE #CH:RETurn
240   END IF
250   IF DMEDIUM_TYPE(#CH)<>1
260     PRINT #0,'This routine only supports floppy disks!!':PAUSE
270     CLOSE #CH:RETurn
280   END IF
290   CLOSE #CH
300   IF LEN(dname$)>10:dname$=dname$(1 TO 10)
310   SELect ON drv_density
320     =1:dname$=dname$&'*D'
330     =2:dname$=dname$&'*H'
340     =3:dname$=dname$&'*E'
350   END SELect
360   FORMAT drv$&dname$
370 END DEFine
```

Usage:

```
REMark Without quotes:
RE_FORMAT flp1_
```

or:

```
REMark With quotes:
RE_FORMAT 'flp2_'
```

**CROSS-REFERENCE**

The name of a medium is set with *FORMAT*. You can read it with *DIR* also.

## 11.70 DMEDIUM_RDONLY

| | |
|---|---|
| Syntax | DMEDIUM_RDONLY [(#channel)] or<br>DMEDIUM_RDONLY (\file) |
| Location | SMSQ/E v2.73+ |

This function returns the value 1 (true) if the the medium on which the specified file or directory is located (or to which the specified channel is open) is write-protected either through hardware or software control. If no parameter is specified, it looks to channel #3 (or #1 if #3 is not open). If the medium can be written to, the value returned is zero.

**NOTE**

This function does not appear to work on Falkenberg hard disk interfaces where it always returns 1.

**CROSS-REFERENCE**

See *DMEDIUM_NAME$* for an example.

## 11.71 DMEDIUM_REMOVE

| | |
|---|---|
| Syntax | DMEDIUM_REMOVE [(#channel)] or<br>DMEDIUM_REMOVE (\file) |
| Location | SMSQ/E v2.73+ |

This function returns the value 1 (true) if the medium on which the specified file or directory is located (or to which the specified channel is open) is a removable hard disk. Otherwise it returns 0 (false). If no parameter is specified, it looks to channel #3 (or #1 if #3 is not open).

**NOTE**

This function does not appear to work on Falkenberg hard disk interfaces where it always returns 1.

**CROSS-REFERENCE**

*DMEDIUM_RDONLY* allows you to check if a disk is write- protected. There do not appear to be any ways in which you can check if any channels are currently open to the medium (ie. whether it is safe to remove the disk), except for listing all currently open channels, see *CHANNELS*.

## 11.72 DMEDIUM_TOTAL

| | |
|---|---|
| Syntax | DMEDIUM_TOTAL [(#channel)] or<br>DMEDIUM_TOTAL (\file) |
| Location | SMSQ/E v2.73+ |

This function returns the number of total sectors available on the medium on which the specified file or directory is located, or to which the specified channel is open. If no parameter is specified, it looks to channel #3 (or #1 if #3 is not open).

**CROSS-REFERENCE**

*DMEDIUM_FREE* allows you to find out the number of sectors which currently do not contain any data on the related medium. *DIR* can also be used to obtain this information. *FORMAT* releases all sectors on a disk, marking any which may be corrupt as unavailable.

## 11.73 DMEDIUM_TYPE

| | |
|---|---|
| Syntax | DMEDIUM_TYPE [(#channel)] or DMEDIUM_TYPE (\file) |
| Location | SMSQ/E v2.73+ |

This function returns a number representing the type of drive on which the specified file or directory is located (or to which the specified channel is open). If no parameter is specified, it looks to channel #3 (or #1 if #3 is not open).

The values currently returned are:

- 0 RAM disk
- 1 Floppy disk drive
- 2 Hard disk drive
- 3 CD-ROM drive

**NOTE**

This function does not appear to work on Falkenberg hard disk interfaces where it always returns 255.

**CROSS-REFERENCE**

See *DMEDIUM_NAME$* for an example.

## 11.74 DNEXT

| | |
|---|---|
| Syntax | DNEXT subdirectory |
| Location | Toolkit II |

This command allows you to move across a directory tree by replacing the current sub-directory with the specified subdirectory in the default data device.

If the default program device is the same as the default data device, then this will also be altered by DNEXT.

If the default destination device is a directory device (ie. if it ends with an underscore), DNEXT also alters the last sub-directory in this (whether or not it points to another drive, or is further down the directory tree).

```
win1_
win1_C_
win1_C_include_
win1_C_objects_
win1_BASIC_
win1_QUILL_
win1_QUILL_letters_
win1_QUILL_translations
win1_secret_
```

The above could be a directory tree on a hard disk. DATA_USE win1_C_ defines win1_C_ as the default directory device, so WDIR will list all of the files in win1_C_.

Assuming that PROGD$='win1_BASIC_' and DESTD$='flp2_C_Include_', entering DNEXT Quill will result in the following:

- DATAD$='win1_Quill_'
- PROGD$='win1_BASIC_'
- DESTD$='flp2_C_Quill_'

**NOTE 1**

DNEXT does not check if there are any files with the given prefix which exist.

**NOTE 2**

DNEXT breaks with error -17 (error in expression) if the parameter is a resident keyword. So append an underscore to the directory name, eg. DNEXT NEW_, or specify the parameter between quote marks (eg. DNEXT 'NEW').

**NOTE 3**

The default devices cannot exceed 32 characters (plus a final underscore) - any attempt to extend them beyond this will result in the error 'Bad Parameter' (error -15).

**CROSS-REFERENCE**

*DUP* moves up the tree, *DDOWN* moves down the tree. *DATAD$* and *DLIST* can be used to find out about the current sub-directory and default devices respectively.

## 11.75 DO

| | |
|---|---|
| Syntax | DO [device_] filename |
| Location | Toolkit II |

This command allows you to execute a set of commands stored in a file (acting as an overlay).

It is intended to perform tasks dictated by a numberless file, which enables you to do many things whilst releasing memory once the tasks have been performed.

DO is actually very similar to the Toolkit II variant MERGE and will ensure that if the given file only contains numberless lines, the channel is closed afterwards.

It does however work just as well as MERGE on numbered files!

A numberless program is basically a set of SuperBASIC lines which do not have any line numbers. These can therefore best be entered with the aid of an editor program. Each line is loaded into the QL with the relevant command, and then executed (one line at a time), as if they had been entered from the command line (#0).

This therefore means that although they can call resident SuperBASIC PROCedures and FuNctions, you can only have in-line structures, such as IF…END IF and SELect ON…END SELect.

Once each line has been executed, it is lost and the memory occupied by that line released.

One advantage for pre JS ROMs is that if you use a numberless file to link resident keywords, such keywords can then be used in the same program which MERGEd the numberless file. For example, if you have a numberless file flp1_resident_bas such as:

```
a=RESPR(12000)
LBYTES flp1_Toolkit,a: CALL a
```

you can then link and use the Toolkit commands in the same program by including a line such as:

```
110 DO flp1_resident_bas
```

**NOTE**

On at least v2.28-v2.49 of Toolkit II, MERGE appears to work much better than DO at executing numberless files. If DO is entered as a direct command, none of the numberless lines are executed (compare MERGE which executes the first line), and if DO is part of a program, only the first line is executed (compare MERGE which executes all of the commands in the numberless file). This is fixed under SMS.

**CROSS-REFERENCE**

Please refer to *MERGE*. SMS allows you to *EXEC*ute a SuperBASIC program, letting it run in the background and perform functions on supplied data using pipes or channels (see *EX*).

## 11.76 DOS_USE

| | |
|---|---|
| Syntax | DOS_USE device$ |
| Location | SMSQ/E for QPC |

DOS_USE may be used to set the name of the DOS device. The name should be three characters long, in upper or lower case.

**Example**

```
DOS_USE mdv
```

The DOS device is renamed MDV.

```
DOS_USE DOS
```

The DOS device is restored to DOS.

```
DOS_USE
```

The DOS device is restored to DOS.

## 11.77 DOS_DRIVE

| | |
|---|---|
| Syntax | DOS_DRIVE drive%, directory$ |
| Location | SMSQ/E for QPC |

This changes the directory the DOS device is connected to.

By default, DOS1_ corresponds to C:\, DOS2_ to D:\ and so on, but the base can be freely chosen in the configuration dialogue or even at runtime:

```
DOS_DRIVE 2, "C:\WINDOWS"
```

will assign DOS2_ to the windows directory on the host's C:\ drive.

```
PRINT DOS_DRIVE$(2)
```

would now return "C:\WINDOWS".

## 11.78 DOS_DRIVE$

| | |
|---|---|
| Syntax | directory$ = DOS_DRIVE$(drive%) |
| Location | SMSQ/E for QPC |

This reads back the currently connected directory of the DOS device.

**Example**

If we continue from the example above for *DOS_DRIVE*, then:

```
PRINT DOS_DRIVE$(2)
```

Will print "C:\WINDOWS".

## 11.79 DOTLIN

| | |
|---|---|
| Syntax | DOTLIN p1, p2, p3, col, x1, y1, x2, y2 |
| Location | HCO |

The command DOTLIN is a variant of LDRAW - it draws a dotted line in the specified colour col from the absolute co-ordinates (x1,y1) to (x2,y2).

The three first parameters are small non-negative integers which specify after how many pixels the line is to be broken (they are known as the periods).

The line is drawn by plotting the first p1 pixels, then leaving a gap of p2 pixels, plotting the next p3 pixels and once again leaving a gap of p2 pixels before recommencing the pattern.

**Examples**

```
DOTLIN 10,10,10,3,40,40,200,60
```

draws a white line from the point (40,40) to the point (200,60) but only for periods of ten pixels.

If a pixel is represented by an asterisk, this would look like this:

```
**********          ********** ...
|-- 10 --||-- 10 --||-- 10 --| ...
```

DOTLIN with the periods 3, 5 and 10:

```
***      **********      ***      **********
|3||-5-||---10---||-5-||3||-5-||---10---|
```

**CROSS-REFERENCE**

All the warnings relevant to SET apply.

## 11.80 DRAW

| | |
|---|---|
| Syntax | DRAW x1,y1 TO x2,y2 ,colour |
| Location | Fast PLOT/DRAW Toolkit |

This command draws a line in absolute co-ordinates on the screen. Any windows and window attributes are ignored. x1 and x2 range from 0 to 511, y1 and y2 from 0 to 255. DRAW uses the screen base address defined with SCRBASE (which enables it to draw on a screen stored in memory as well as the currently visible screen. It is therefore compatible with QL emulators and Minerva's dual screen mode, although it cannot support higher resolutions). The range for the colour parameter is 0..7, other values give odd results without being dangerous.

**Example**

Here is a procedure which draws a line given in polar co-ordinates. A point in a polar system is specified by a radius and angle.

```
170 DEFine PROCedure POLAR_DRAW (r1,phi1,r2,phi2,col)
180   REMark less precise but fast version
190   LOCal r,phi,r_old,phi_old,dr,dphi
200   LOCal x1,x2,y1,y2,dist
210   r_old=r1: phi_old=phi1
220   r=r1: phi=phi1
230   x1=1.35*r_old*SIN(phi_old+PI/2)+255
240   y1=r_old*COS(phi_old+PI/2)+127
250   dist=(r1+r2)/2 * (phi1+phi2)/PI
260   IF dist==0 THEN RETurn
270   dr=(r2-r1)/dist: dphi=(phi2-phi1)/dist
280   REPeat Drawing
290     IF r>=r2 AND phi>=phi2 THEN EXIT Drawing
300     r=r_old+dr: phi=phi_old+dphi
310     x2=1.35*r*SIN(phi+PI/2)+255
320     y2=r*COS(phi+PI/2)+127
```

(continues on next page)

(continued from previous page)

```
330     DRAW x1,y1 TO x2,y2 ,col
340     r_old=r: phi_old=phi
350     x1=x2: y1=y2
360   END REPeat Drawing
370 END DEFine POLAR_DRAW
```

```
POLAR_DRAW 0,0 TO 100,8*PI ,7
```

draws an archimedic spiral and these few lines create a polar pattern:

```
10 SCLR 0
20 FOR a=0 TO 50 STEP 10
30 POLAR_DRAW a,0 TO a,2*PI ,7
40 POLAR_DRAW 0,PI*a/25 TO 50,PI*a/25, 7
50 END FOR a
60 REFRESH
```

**NOTE**

DRAW is specific to the resolution of 512x256 pixels. It can however be used to draw on Minerva's second screen by using the command SCRBASE SCREEN(#3) (assuming that #3 is open on the second screen).

**CROSS-REFERENCE**

*PLOT* plots a pixel, *SCLR* clears the screen and *REFRESH* makes the screen pointed to by *SCRBASE* visible. See also *DOTLIN* and the other variant of *DRAW*.

---

## 11.81 DRAW

| | |
|---|---|
| Syntax | DRAW [#ch,] x2,y2 |
| Location | DRAW (DIY Toolkit - Vol G) |

This command draws a line in absolute co-ordinates on the screen with reference to the specified window channel (if any - default #1). The line is drawn from the last point plotted with the PLOT command from the same toolkit, to the point specified by x2,y2. This is quicker than using the SuperBASIC LINE command and unlike other similar commands, it will support the current INK colour and OVER mode.

<CTRL><F5> will pause the line drawing and it will even work in MODE 4, 8 and 12 (on the THOR XVI, provided that you have v1.6+). The main limitation on this command is that the line must appear fully within the specified window, so x2 and y2 cannot exceed the width or height of the specified window (in pixels) nor be less than zero.

**NOTE**

Although DRAW will work wherever the screen base is located, it assumes that a line of pixels takes 128 bytes - it will not therefore currently work on higher resolutions.

**CROSS-REFERENCE**

See the other variant of*DRAW*. See also *PLOT*. *LINE* is much more flexible.

## 11.82 DROUND

| | |
|---|---|
| Syntax | DROUND (d, x) |
| Location | TRIPRODRO |

The function DROUND will return the floating point number x rounded to d decimal digits, counted from the left of the number. DROUND rounds the last digit up or down depending on the next digit (ignoring any others).

**Example**

```
DROUND(3, 1234.56) = 1230
DROUND(4, 1234.56) = 1235
```

**CROSS-REFERENCE**

*PROUND* rounds to a given precision.

## 11.83 DUP

| | |
|---|---|
| Syntax | DUP |
| Location | Toolkit II |

This command strips off the last part of the default data device, thus moving up the directory tree. If the default program device is the same as the default data device, then this will also be altered by DUP. If the default destination device is a directory device (ie. if it ends with an underscore), DUP also alters this (whether or not it points to another drive).

```
win1_
win1_C_
win1_C_include_
win1_C_objects_
win1_BASIC_
win1_QUILL_
win1_QUILL_letters_
win1_QUILL_translations
win1_secret_
```

If DATAD$ is win1_, DDOWN Quill moves down to win1_Quill_, whilst DUP will move DATAD$ back up to win1_.

If DATAD$ is win1_Quill_letters_secret_, three DUPs will change it back to win1_.

**NOTE**

It is not possible to move beyond the name specifying the actual device to be used. In the above example, this is the named root, win1_.

**CROSS-REFERENCE**

*DATA_USE* allows you to set the absolute directory root, *DDOWN* goes down the tree, and *DNEXT* skips from branch to branch. *DATAD$* returns the current default data device ie. the device name plus the current sub-directory.

CHAPTER

TWELVE

# KEYWORDS E

## 12.1 EASTER

| | |
|---|---|
| Syntax | EASTER (year%) where year% >= 1583 |
| Location | Math Package |

This function calculates the date of Easter Sunday for any given year after 1583 (when the Gregorian calender was introduced by Pope Gregory XIII to replace the Julian calender of Julius Ceasar which had been in use since 46 BC). EASTER returns the date as a floating point number, where the day is the integer part of the number and the month is given by the digits following the floating point, eg. PRINT EASTER(1993) shows 11.4 (April, 11th)

**Example**

Easter Sunday is used as a basis to fix other clerical days, so if two years have Easter Sunday on the same day, the other holy dates are identical, too:

```
100 INPUT "Year=";year
110 east1=EASTER(year)
120 FOR y=year+1 TO 32767
130   east2=EASTER(y)
140   IF east1=east2 THEN
150     PRINT "Next Easter Sunday on"!east1;". is in"!y;"."
160     EXIT y
170   END IF
180 END FOR y
```

**NOTE**

EASTER does not return the correct value on SMSQ/E for some reason.

**CROSS-REFERENCE**

*GREGOR*

## 12.2 ED

| | |
|---|---|
| Syntax | ED [#ch,] [start_line] |
| Location | Toolkit II |

This command invokes Toolkit II's full-screen editor. This provides powerful facilities for editing a SuperBASIC program loaded in memory and forms a useful alternative to the QL's standard EDIT and AUTO commands.

ED will list the current SuperBASIC program from its first line (or from the specified start_line) onwards in the given channel (default #2). If the specified channel (#ch) is not a console con_ channel, then an error -15 (Bad Parameter) will be reported. If any lines are too long to fit in the specified window, they are wrapped round onto the next line, with this 'continuation line' indented in order to differentiate from other program lines. It does however make sense to use the widest possible window to avoid wrapping of lines.

Once a window-full of the listing is shown, ED will activate the cursor in the window and you can then move up or down through the listing by using the up and down cursor keys. The left and right cursor keys will move across the listing lines (and even 'blank' space where the actual program lines do not appear).

Any attempt to alter a line (eg. to delete a character) will activate that line, in which case it will be shown in inverse colours. Any attempt to move the cursor off that line (or pressing <ENTER>) will tell ED to accept the alterations and de-activate that line.

If the line is not acceptable to the SuperBASIC parser, then a 'Bad Line' error will be generated in #0 and the line re-activated.

If you press the Break key or <ESC> whilst a line is active, it will be de-activated and returned to its original state. If no line is active, <ENTER> will insert a new line number half-way (if possible) between the number of the line on which the cursor is situated and the next line number. If there is no room for an additional line between the two program lines, <ENTER> will be ignored.

If on the other hand, there is a gap of 20 or more (or there are no further program lines), the new line number will be the current line number plus 10.

Another way of creating new lines is to amend the line number of the current line. If you do this, a new line with the amended line number will be inserted (overwriting any existing line) and the current line will remain the same (the cursor remains on the same line). This enables you to copy lines from one part of a program to another.

By way of further assistance to the SuperBASIC programmer, ED can work in two modes - Overwrite Mode and Insert Mode. The latter is the default, in which case any characters typed will activate the current line and insert them at the current cursor position.

In Overwrite Mode, any characters typed will activate the current line and replace the characters under the cursor.

A line can be deleted either by using <CTRL><ALT><←> (except on SMS where you must use <CTRL><←>) or by deleting all of the visible characters in a line. If you delete everything but the line number, then the line pointed to by that line number will be deleted.

There are several other keys available which make editing a SuperBASIC program much easier than under EDIT. The keys available from within the standard ED are listed on the next page.

**NOTE 1**

Avoid ED #0 if possible.

**NOTE 2**

ED is likely to cause various problems if used from within a SuperBASIC program.

**NOTE 3**

Any attempt to create SuperBASIC lines which are longer than 32766 characters may crash SuperBASIC.

**NOTE 4**

If a program contains a line 32767, this may upset ED when you are editing the end of the program.

**NOTE 5**

ED is not very helpful when there is no program actually in memory, as it starts off with a blank screen and you have to type the whole line, including line number (even if you passed a line number with the command).

**NOTE 6**

As from SMS v2.58 you can use ED ERLIN to edit the line which has caused an error. We are not certain if this works on other implementations.

**NOTE 7**

Any attempt to ED a line number greater than 32757 can cause problems (on some versions ED creates negative line numbers, on others you cannot see the line being edited). SMSQ/E v2.85 (at least) does not have these problems, but see Note 4 above.

**ED Special Key Presses**

The keys available in ED are:

| Key Press | Action |
|---|---|
| <ENTER> | Create new line, unless line is active, in which case this tells ED to accept alterations to the line and de-activate it. |
| <ESC> | Leave ED - control returns to #0 unless line is active, in which case this de-activates line without altering it. |
| <CTRL><SPAC | See <ESC>. |
| <TAB> | Move to the right by multiples of eight. |
| <SHIFT><TAB | Move to the left by multiples of eight. |
| ↑ | Move up one line. |
| <ALT> ↑ | Scroll up a line (cursor remains still, text moves down). |
| <SHIFT> ↑ | Scroll up one page (cursor remains still). |
| ↓ | Move down one line. |
| <ALT> ↓ | Scroll down a line (cursor remains still, text moves up). |
| <SHIFT> ↓ | Scroll down one page (cursor remains still). |
| → | Move right one character. |
| <CTRL> → | Delete character under cursor (line becomes active). |
| ← | Move left one character. |
| <CTRL> ← | Delete character to left of cursor (line becomes active). |
| <CTRL><ALT> ← | Delete line under cursor (not under SMS). |
| <SHIFT><F4> | Switch between overwrite and insert mode. |

SMS adds the following additional keys:

| Key Press | Action |
|---|---|
| <SHIFT> ← | Move left one word. |
| <ALT> ← | Move to start of line. |
| <CTRL>< ← | Delete word to left of cursor (line becomes active). |
| <CTRL>< ← | Delete from cursor to start of line (line becomes active). |
| <SHIFT> → | Move right one word. |
| <ALT> → | Move to end of line. |
| <CTRL>< → | Delete word under cursor (line becomes active). |
| <CTRL>< → | Delete from cursor to end of line (line becomes active). |
| <CTRL> ← | Delete whole line under cursor. |
| <SHIFT>< | Stuff the currently activated line into the Hotkey buffer so that this can be later recalled with <ALT><SPACE>. **Note** this will only work if the Hotkey system is active (see HOT_GO). For this you need v2.58+. |

**SMS NOTES**

Oddly, the SuperBasic interpreter allows you to enter a line which is beyond the permitted range of line numbers, for example, enter as a direct command:

```
40000 PRINT 'This should not be accepted'
```

No error is reported, and the line is executed as if it had been entered without a line number!

SMS also suffers with problems if you edit a long line at the bottom of a window, so that as you type in more text for the line, the program line extends below the bottom of the window. 'Invalid Syntax' is printed over and over in #0, crashing the computer. This was improved in v2.71 but still has not been totally fixed.

The keying <CTRL> → clashes with the key used by early versions of the program MasterBasic (by Ergon Software) which is used to move between occurrences of an object which has been searched for in the program. This has been resolved in v1.46+ of the program.

If you try to use ED on #2 and this is not open, then SMS will use #0 (if this is not open, it will open a default window #0). This is useful for SBASICs which may be started with only one channel open (an input channel).

Another useful feature implemented on SMS is that as from v2.69, if you enter the command ED without any parameters, this has one of two effects. If you have not previously used ED, this edits the start of the program (as on all other versions). However, if you have previously used ED, the line which is shown at the top of #2 is the line which was at the top of the window when you left ED previously - this can therefore be useful when testing a section of the program.

**CROSS-REFERENCE**

Please also refer to *AUTO* and *EDIT* which are replaced by this command.

## 12.3 EDIT

| | |
|---|---|
| Syntax | EDIT [start_number] [,step] |
| Location | QL ROM |

This command allows you to enter the SuperBASIC line editor in order to alter a SuperBASIC program loaded in memory. It will automatically create line numbers in the command line (#0) to assist in entering SuperBASIC programs, in much the same way as AUTO. EDIT would normally only be entered as a direct command (although you can include it in a program line, the line numbers will not be generated until the program has finished its work).

Once entered, you will be presented with the first line start_number (default 100) - if this line already exists in the program, then the existing line will be presented. Otherwise, you will only see the current line number.

Pressing the up and down arrow keys will move you to the previous line or the next line (respectively) in the program, although if there is no previous (or next) line, then you will exit the EDIT mode. However, if you press the Enter key, if step is specified (default 0), this will act in the same way as AUTO. However, if step is not specified, you will leave EDIT mode.

The main advantage of using EDIT over ED is how EDIT handles the screen. If the program has not been previously EDITed (or a PROC/FN Cleared message has been displayed) then EDIT will show a section of the current program in #2 when you move off the line currently being EDITed with the cursor keys or <ENTER>. This section will have the line which was just EDITed as the top line and will go on to fill #2 with additional lines of the program. However, if the program has already been EDITed and the PROC/FN Cleared message has not been displayed, then EDIT will not affect the display on screen (other than showing parts of the program in #0) until you EDIT a line which is within the range of lines which were previously being EDITed.

This range of lines is actually slightly bigger than the lines which would have been displayed in #2, going from an invisible top line (the line above the displayed line) to an invisible bottom line (the line below the displayed line). Now, this can be quite useful when searching a program for some text or deciding where to copy a section of the program to, or even to line up characters on screen when the program has been RUN.

The listing which last appeared on #2 is represented as:

```
110 PAPER 0:INK 4:CLS(Invisible Top Line)
--------------------------------------------
120 PRINT 'A PROGRAM'(Displayed Lines)
130 PRINT 'TO GET YOUR NAME'
140 INPUT \\'ENTER YOUR NAME';name$
150 PRINT \\
160 PRINT 'HELLO'!name$
--------------------------------------------
170 PRINT \\"I'M YOUR COMPUTER"(Invisible bottom Line )
```

**NOTE 1**

You cannot set an absolute step value of zero - omit this parameter to achieve the same result!

**NOTE 2**

On non-Minerva ROMs EDIT uses the same routine as RENUM to check its parameters, which means that you can specify a start_line and an end_line, although they do nothing. For example:

```
EDIT 100 TO 1000;1000,20
```

would create lines 1000, 1020, 1040, ....

**NOTE 3**

The maximum line number is 32767. Both start_number and step should be integers - if they are not, they will be rounded to the nearest integer (compare INT).

**NOTE 4**

Additional keys are available for editing on Minerva (see INPUT).

**NOTE 5**

EDIT can give problems if it is issued after breaking into a program which was in the middle of a PROCedure or FuNction at the time.

On non-Minerva ROMs, this is likely to produce a 'not implemented' error and the wrong line. Press Break and try again do not try to edit the line. On Minerva ROMs (pre v1.97) this is compounded by the fact that Minerva tends to try to run the program again.

Sometimes you are lucky and Minerva tries to jump to a non-existent line number before presenting you with the desired line. Unfortunately, EDIT is never really safe in this context, and you should either type CLEAR before EDIT or use ED.

**NOTE 6**

On pre Minerva ROMs SuperBASIC is liable to lock up if you try to EDIT a line after trying to call a PROCedure/FuNction which was defined at the end of the program, but had been deleted.

**SMS NOTES**

On SMS the EDIT command is exactly the same as ED.

**CROSS-REFERENCE**

*AUTO* is very similar, especially where *STEP* is specified. *DLINE* deletes program lines. *INPUT* contains details of the available keypresses for cursor navigation. *ED* provides a different means of editing a SuperBASIC program.

```
PRINT PEEK_W(\\HEX('9C'))
```

returns the line number of the invisible top line which was last *EDIT*ed (except on SMS).

```
PRINT PEEK_W(\\HEX('9E'))
```

returns the line number of the bottom line in #2 which was last *EDIT*ed (except on SMS).

## 12.4 EDITF

| | |
|---|---|
| Syntax | EDITF ([#ch,] {default \| default$} [,maxlen%]) |
| Location | Turbo Toolkit |

This function is similar to *EDLINE$*. However, *EDITF* is intended solely for asking the user to enter a floating point number. The specified default (which may be given as a number or a string) is printed at the current text cursor position in #ch (default #1) and allows you to edit it. The parameter maxlen% dictates the maximum number of characters allowed (this defaults to the amount set when the Turbo Toolkit is configured). The edited result is returned when <ENTER> is pressed. If the string contains a nonsensical value when <ENTER> is pressed, a warning beep is sounded.

**NOTE**

On non-SMS machines, a buffer full error could be reported if an attempt was made to enter a string longer than 118 characters, or the length of the longest SuperBASIC line listed or edited to date, whichever is longer.

**CROSS-REFERENCE**

See *EDLINE$*. *EDIT%* and *EDIT$* are also useful.

## 12.5 EDIT%

| | |
|---|---|
| Syntax | EDIT% ([#ch,] {default \| default$} [,maxlen%]) |
| Location | Turbo Toolkit |

This function is the same as *EDITF*, except that only integer values are acceptable.

**CROSS-REFERENCE**

See *EDITF*.

## 12.6 EDIT$

| | |
|---|---|
| Syntax | EDIT$ ([#ch,] default$ [,maxlen%]) |
| Location | Turbo Toolkit |

This function is similar to *EDLINE$*. It operates in the same way as *EDITF*, except that any string of characters can be edited, rather than being restricted to a number.

**CROSS-REFERENCE**

See *EDITF*.

## 12.7 EDLINE$

| | |
|---|---|
| Syntax | EDLINE$ (#ch, maxlen%, edit$) |
| Location | EDLINE (DIY Toolkit Vol E) |

The function EDLINE$ prints edit$ at the current text cursor position in #ch (there is no default channel) and allows you to edit it. A maximum length of maxlen% characters is allowed. The edited result is returned. Unlike INPUT, EDLINE$ will not finish with <UP> or <DOWN> but only after <ENTER> and <CTRL><SPACE> (also <ESC> on Minerva). Instead <UP> and <DOWN> move the cursor to the start

and end of the string respectively; apart from that the usual keys for editing are used: <CTRL><LEFT> deletes the character to the left of the cursor, <CTRL><RIGHT> the character under the cursor.

**Example**

```
100 CLS
110 REPeat ask_name
120 PRINT \"Please enter your name: ";
130 Name$ = "Billy the Kid"
140 Name$ = EDLINE$(#1,40,Name$)
150 PRINT "Your name is '";Name$;"' (y/n)? ";
160 ok$ = EDLINE$(#1,1,"y")
170 IF ok$ INSTR "yY" THEN EXIT ask_name
180 PRINT "Try again..."
190 END REPeat ask_name
200 PRINT "Hello,"!Name$;"!"
```

**NOTE**

You need a special version of EDLINE$ to work correctly on Minerva and SMS. This version is included with the DIY Toolkit package.

**CROSS-REFERENCE**

*EDLINE$* can be used to input numbers but you have to ensure that the entered text can be successfully coerced to a number, see *CHECK%* and *CHECKF* for that. *EDIT$* is similar. Other routines for human input are for example: *INPUT*, *INKEY$*, *ASK* and *REPLY*.

## 12.8 EL

| | |
|---|---|
| Syntax | EL |
| Location | Beuletools |

This function returns the control codes needed to switch on the NLQ ( near letter quality) font on an EPSON compatible printer:

```
PRINT EL
```

is the same as:

```
PRINT CHR$(27)&"x"&CHR$(1).
```

**CROSS-REFERENCE**

*NORM*, *BLD*, *DBL*, *ENL*, *PRO*, *SI*, *NRM*, *UNL*, *ALT*, *ESC*, *FF*, *LMAR*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 12.9 ELIS

| | |
|---|---|
| Syntax | ELIS (keyword$) |
| Location | TinyToolkit |

This function will return the machine code start address of the specified resident keyword if it is recognised by SuperBASIC. If the keyword is unknown, then the function will generate a Not Found error.

**CROSS-REFERENCE**

See *KEY_ADD*, *FLIS* and *CODEVEC*. Compare *FIND* and *LOOKUP%*.

## 12.10 ELLIPSE

| | |
|---|---|
| Syntax | ELLIPSE [#ch,] x,y,radius,ratio,ecc $^{*}$[;$x^{i}$,$y^{i}$,$radius^{i}$,$ratio^{i}$,$ecc^{i}$]$^{*}$ |
| Location | QL ROM |

Both the ELLIPSE and CIRCLE commands perform exactly the same function. We have however decided to split them, since if users adopt the habit of using ELLIPSE to draw ellipses and CIRCLE to draw circles, this will help users understand SuperBASIC programs much more easily.

This command allows you to draw an ellipse in the current INK colour of the given radius with its centre point at the point (x,y).

The ratio affects the difference between the major axis and the minor axis - the greater the ratio, the greater the difference between the two.

The major (y) axis is specified by the parameter radius, whereas the minor (x) axis is calculated by radius*ratio which therefore means that if ratio>1, the major axis will become the (x) axis (if you see what we mean!).

Ecc defines the angle at which the ellipse will be drawn. This is measured in radians and forms the anti-clockwise angle between a vertical line drawn through the origin of the ellipse and the major axis. Thus, ecc=PI/4 draws an ellipse at an angle of 45 degrees.

The actual positioning and size of the ellipse will depend upon the scale and shape of the specified window (default #1).

The co-ordinates are calculated by reference to the graphics origin, and the graphics pointer will be set to the centre point of the last ellipse to be drawn on completion of the command. If any parts of the ellipse lie outside of the specified window, they will not be drawn (there will not be an error).

If the parameters ratio and ecc are omitted, this command has exactly the same effect as CIRCLE. This command will actually allow you to draw multiple ellipses by including more sets of parameters. Each additional set must be preceded by a semicolon (unless the preceding ellipse uses five parameters). This means for example, that these all perform the same action:

```
ELLIPSE 100,100,20,1,2,50,50,20
ELLIPSE 100,100,20,1,2; 50,50,20
ELLIPSE 100,100,20,1,2: CIRCLE 50,50,20
```

Although the FILL command will allow you to draw filled ellipses on screen (in the current ink colour), you will need to include a FILL 1 statement prior to each ellipse if they are to appear independently on screen (this cannot be achieved when using this command to draw multiple ellipses).

If this rule is not followed, then any points which lie on the same horizontal line (even though they may be in different ellipses) will be joined.

**Example**

Try the following for an interesting effect:

```
100 MODE 8
110 WINDOW 448,200,32,16:PAPER 0:CLS
120 SCALE 100,0,0
130 INK 4:OVER -1
140 REPeat loop
150   FOR ang=0 TO PI*2-(PI*2/20) STEP PI*2/20
160     FILL 1:ELLIPSE 70,50,40,.5,ang
170     FILL 1:ELLIPSE 70,50,40,.5,ang
180   END FOR ang
190 END REPeat loop
```

**NOTE**

On all ROMs other than Minerva v1.89+, very small ellipses and very large ones can cause problems. Try:

```
ELLIPSE 80,80,80,6,1
```

on a non-Minerva machine for a laugh.

Unfortunately, Lightning SE (v2.11) still contains this bug and will bring it back!

**CROSS-REFERENCE**

Please refer to *CIRCLE*, *ELLIPSE_R*, *ARC*, *LINE* and *POINT*.

## 12.11 ELLIPSE_R

| | |
|---|---|
| Syntax | ELLIPSE_R [#ch,] x,y,radius,ratio,ecc $^{*}$[;$x^{i}$,$y^{i}$,$radius^{i}$,$ratio^{i}$,$ecc^{i}$]$^{*}$ |
| Location | QL ROM |

This command draws an ellipse relative to the current graphics cursor. See *ELLIPSE* above!

**CROSS-REFERENCE**

Please refer to *ARC_R* and *CIRCLE_R*.

## 12.12 ELSE

| | |
|---|---|
| Syntax | ELSE $^{*}$[:statements]$^{*}$ |
| Location | QL ROM |

This command forms part of the IF…END IF structure and allows you to take alternative action if the condition contained in the IF statement proves to be false.

**CROSS-REFERENCE**

See *IF* for more details.

## 12.13 END

| | |
|---|---|
| Syntax | END ... |
| Location | QL ROM |

This keyword forms part of the structures: END WHEN, END SELect, END IF, END REPeat, END FOR and END DEFine As such, it cannot be used on its own within a program - this will cause a 'bad line' error.

**CROSS-REFERENCE**

Please refer to the individual structure descriptions below for more details.

## 12.14 END DEFine

| | |
|---|---|
| Syntax | END DEFine [name] |
| Location | QL ROM |

This command marks the end of the DEFine PROCedure and DEFine FuNction SuperBASIC structures, and has no meaning on its own. You may if you wish, place the name of the PROCedure or FuNction after END DEFine to help make the SuperBASIC program more readable - this will however have no effect on how the command is treated by the interpreter, which will still take the next END DEFine as the end of the current definition block (even if it is followed by a different name).

The interpreter will jump out of a definition block whenever it meets a RETurn statement. It will also jump out of a DEFine PROCedure definition when it meets an END DEFine statement. This does of course mean that END DEFine can be used in the middle of a PROCedure to force a return to the calling statement - however, this can cause other problems and a RETurn should be used, with END DEFine only appearing at the very end of the definition block.

On the other hand, the interpreter can only jump out of a DEFine FuNction definition if it meets a RETurn - if the interpreter comes across an END DEFine in such situations, it will report the error 'Error In Expression'. On SMS the error 'RETurn not in Procedure or Function' is reported. If the definition block is not actually being used, but the interpreter is working its way through the program, when a DEFine PROCedure or DEFine FuNction statement is met, the interpreter will search for the next END DEFine, and having found one, will resume the program at the next statement.

This does however mean, that unless an in-line DEFine structure is being used, if this command is missing, the interpreter will carry on searching through the program and may just stop without an error if END DEFine does not appear anywhere in the program after the initial DEFine PROCedure (or DEFine FuNction).

**Example 1**

The above rules mean that the following example will work under SuperBASIC, but is extremely inefficient and difficult to decode:

```
10 FOR i=1 TO 100
20   PRINT power(i)
30   DEFine FuNction power(x)
40     RETurn 2^x
```

(continues on next page)

(continued from previous page)

```
50   END DEFine
60 END FOR i
```

**Example 2**

See if you can work out why the following program goes wrong:

```
100 FOR i=1 TO 100
110   PRINT power(i)
120   DEFine FuNction power(x)
130     DEFine FuNction base
140       RETurn 2
150     END DEFine base
160     RETurn base^x
170   END DEFine power
180 END FOR i
```

If you are having trouble, try inserting:

```
155 PRINT 'Program line 155:';x
```

**NOTE**

END DEFine need not appear in an in-line definition statement, except under SMS.

**SMS NOTE**

Checks are made on a program before it is run, and so if an END DEFine statement is missing, this will be reported as an error ('Incomplete DEFine clause'). SMS's improved interpreter will report the error 'Misplaced END DEFine' if END DEFine does not mark the end of a DEFine PROCedure or DEFine FuNction block.

**CROSS-REFERENCE**

Please see *DEFine PROCedure* and *DEFine FuNction*. Other SuperBASIC structures are *SELect ON*, *IF*, *REPeat*, WHEN XXX and *FOR*.

## 12.15 END FOR

| | |
|---|---|
| Syntax | END FOR loop |
| Location | QL ROM |

This command marks the end of the FOR..END FOR SuperBASIC structure with the same loop name, and has no real meaning on its own. When the interpreter meets this statement, it then looks at the stack to see if a related FOR command has already been parsed.

If not, then the error 'Not Found' will be reported, however, if such a FOR loop has been parsed, the interpreter will fetch the end parameter and if the loop is not yet at this value, then step is added to loop and control returned to the statement following FOR.

If however loop is already at the end value, control passes to the statement following END FOR.

The second variant is only available under SMS, where the interpreter presumes that if no loop name is specified, the programmer means the interpreter to return control to the most recent FOR statement (if the loop is not at its final value).

When an EXIT loop is found, the interpreter will search for the relative END FOR loop, and if found, will resume program flow at the next statement.

Under SMS, neither EXIT nor END FOR need have a loop identifier, and therefore EXIT will simply cause the program to jump to the statement after the next END FOR command (if no loop is specified).

This does however mean, that except under SMS, unless an in-line FOR structure is being used, if this command is missing, the interpreter will carry on searching through the program and may just stop without an error if END FOR loop does not appear anywhere in the program.

**NOTE**

END FOR need not appear in an in-line FOR statement.

**SMS NOTE**

SMS will report 'unable to find an open loop' if the interpreter comes across an END FOR command (without a loop variable name) without a corresponding open FOR loop. If the interpreter comes across an END FOR command (with a loop variable name) without a corresponding open FOR loop the error 'undefined loop control variable' is reported.

**CROSS-REFERENCE**

Please see *FOR*. Compare *NEXT* and *EXIT*. Other SuperBASIC structures are: *DEFine PROCedure*, *DEFine FuNction*, *SELect ON*, *IF*, *REPeat*, and WHEN XXX.

## 12.16 END IF

| | |
|---|---|
| Syntax | END IF |
| Location | QL ROM |

This command marks the end of the IF..END IF SuperBASIC structure, and has no meaning on its own.

When the interpreter finds an IF condition statement it then evaluates the condition and carries out certain commands depending on whether the condition was true or false.

Having carried out those commands, the interpreter then looks for a related END IF command, and will pass control onto the statement following END IF.

This does however mean, that except under SMS, unless an in-line IF structure is being used, if this command is missing, the interpreter will carry on searching through the program and may just stop without an error if END IF does not appear anywhere in the program.

**NOTE 1**

END IF need not appear in an in-line IF statement.

**NOTE 2**

All ROMs (except for Minerva v1.93+ or SMS) can get mixed up with multiple in-line IF..END IF structures - see IF.

**SMS NOTE**

Checks are made on a program before it is run, and so if an END IF statement appears without a corresponding IF command, the error 'Misplaced END IF' is reported.

**CROSS-REFERENCE**

Please see *IF*. Other SuperBASIC structures are: *DEFine PROCedure*, *DEFine FuNction*, *SELect ON*, *REPeat*, *FOR*, and WHEN XXX.

## 12.17 END REPeat

| | |
|---|---|
| Syntax | END REPeat identifier or<br>END REPeat [identifier]SMS only |
| Location | QL ROM |

This command marks the end of the REPeat…END REPeat SuperBASIC structure with the same identifier, and has no meaning on its own.

When the interpreter meets this statement, it then looks at the stack to see if a related REPeat command has already been parsed. If not, then the error 'Not Found' will be reported, however, if such a REPeat identifier has been parsed, the interpreter will force the program to loop around and return control to the statement following REPeat.

Under SMS there is no need to specify the identifier on the END REPeat statement, in which case, the interpreter will presume that this is the end of the last REPeat loop to have been encountered.

When an EXIT identifier is found, the interpreter will search for the relative END REPeat identifier (or under SMS the next END REPeat command), and if found, will resume program flow at the next statement.

This does however mean, that except under SMS, unless an in-line REPeat structure is being used, if this command is missing, the interpreter will carry on searching through the program and may just stop without an error if END REPeat identifier (or END REPeat under SMS) does not appear anywhere in the program.

**NOTE**

END REPeat need not appear in an in-line REPeat statement.

**SMS NOTE**

SMS will report 'unable to find an open loop' if the interpreter comes across an END REPeat command (without a loop identifier) without a corresponding open REPeat loop. If the interpreter comes across an END REPeat command (with a loop identifier) without a corresponding open REPeat loop the error 'undefined loop control variable' is reported.

**CROSS-REFERENCE**

Please see *REPeat*.

*NEXT* loop_variable is practically the same although see *EXIT*. Other SuperBASIC structures are: *DEFine PROCedure*, *DEFine FuNction*, *SELect ON*,*IF*, *FOR*, and WHEN XXX.

## 12.18 END SELect

| | |
|---|---|
| Syntax | END SELect |
| Location | QL ROM |

This marks the end of the SELect ON…END SELect SuperBASIC structure, and has no meaning on its own. When the interpreter has found a match for the value of the variable, it performs a series of commands, and then looks for the end of the block marked with END SELect.

This means that except under SMS, unless an in-line SELect ON structure is being used, if this command is missing, the interpreter will carry on searching through the program and may just stop without an error if END SELect does not appear anywhere in the program.

**NOTE 1**

END SELect need not appear in an in-line SELect ON statement.

**NOTE 2**

Under SMS, if END SELect appears in an in-line SELect ON statement, if any commands appear after END SELect on the same line, an error will be reported.

**SMS NOTE**

Checks are made on a program before it is run, and so if an END SELect statement is missing, this will be reported as an error ('Incomplete SELect clause'). SMS's improved interpreter will report the error 'Misplaced END SELect' if END SELect does not mark the end of a SELect ON definition block.

**CROSS-REFERENCE**

Please see *SELect ON*. Other SuperBASIC structures are *DEFine PROCedure*, *DEFine FuNction*, *IF*, *REPeat*, WHEN XXX and *FOR*.

## 12.19 END WHEN

| | |
|---|---|
| Syntax | END WHEN |
| Location | QL ROM (post JM) |

This marks the end of the SuperBASIC structures: WHEN ERRor and WHEN condition … END WHEN, and has no meaning on its own. When the program is first run, the interpreter marks the start of this structure and then (unless it is an in-line structure) looks for the end of the block marked with END WHEN.

This means that if this statement is missing, except under SMS, the interpreter will carry on searching through the program and may just stop without an error if END WHEN does not appear anywhere in the program.

**NOTE**

END WHEN need not appear in a single line WHEN or WHEN ERRor statement, eg:

```
100 WHEN a>4:PRINT 'a>4'.
```

**SMS NOTES**

Checks are made on a program before it is run, and so if an END WHEN statement is missing, this will be reported as an error.

SMS's improved interpreter will also report the error 'Misplaced END WHEN' if END WHEN does not mark the end of a WHEN ERROR definition block.

**CROSS-REFERENCE**

Please see *WHEN ERRor* and *WHEN condition*. Other SuperBASIC structures are *DEFine PROCedure*, *DEFine FuNction*, *IF*, *REPeat*, *SELect* and *FOR*.

## 12.20 END_CMD

| | |
|---|---|
| Syntax | END_CMD |
| Location | Turbo Toolkit |

This marks the end of a numberless file of direct commands for use with the MERGE command. This command should be entered on its own as the last line of the numberless file. It overcomes the problem explained in NOTE 1 of MERGE.

**CROSS-REFERENCE**

Please see *MERGE*. *DO* is also useful for executing such files.

## 12.21 END_WHEN

| | |
|---|---|
| Syntax | END_WHEN |
| Location | Turbo Toolkit |

This marks the end of the Turbo structure equivalent to the SuperBASIC WHEN ERRor structure. END_WHEN has no meaning on its own and should only be used within Turbo compiled programs.

**CROSS-REFERENCE**

Please see *WHEN ERRor*

## 12.22 ENV_DEL

| | |
|---|---|
| Syntax | ENV_DEL name$ |
| Location | Environment Variables |

This command is used to remove a specified environment variable. Please note that the name of the environment variable is case sensitive. If an empty string is passed as the argument, then an error will be reported.

**Example**

A boot program may specify where the files for the main program are stored and then pass it to subsequently called programs with. Once the program has finished, the environment variable may be deleted.

```
1000 source$='win1_PROGS_utils_'
1010 SETENV "PROGLOC="&source$
1020 EXEC_W source$&'main_exe'
1030 ENV_DEL "PROGLOC"
```

**CROSS-REFERENCE**

Please see SETENV.

## 12.23 ENV_LIST

| | |
|---|---|
| Syntax | ENV_LIST [#ch] |
| Location | Environment Variables |

This command lists all currently active environment variables to the specified channel (default #1).

**CROSS-REFERENCE**

Please see SETENV.

## 12.24 ENL

| | |
|---|---|
| Syntax | ENL |
| Location | Beuletools |

This function returns the control codes needed to switch on double width on an EPSON compatible printer:

```
PRINT ENL
```

is the same as:

```
PRINT CHR$(27)&"W"&CHR$(1)
```

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*, *DBL*, *PRO*, *SI*, *NRM*, *UNL*, *ALT*, *ESC*, *FF*, *LMAR*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 12.25 EOF

| | |
|---|---|
| Syntax | EOF [(#ch)] |
| Location | QL ROM |

This is a logical function which actually has two uses in SuperBASIC. If no channel number is specified, then PRINT EOF will return 1 unless the current program contains some DATA lines which have not yet been READ. This is therefore useful to create programs which can handle any amount of data. However, if a channel number is specified, for example PRINT EOF(#1), then zero will be returned unless the given

channel is linked to a file and the file pointer is at (or beyond) the end of that file (ie. whether or not there is data to be read from that channel).

**Example**

Two simple programs to retrieve an address from a given name (the full name must be given on input). The first of these has the data stored in the program, whereas the second has it stored on a file called flp1_address_data:

```
100 RESTORE
110 MODE 4
120 OPEN #3,con_448x200a32x16:BORDER#3,1,2:PAPER#3,0:INK#3,7
130 INPUT #3,'Input name to look for:'!search$
140 REPeat loop
150   IF EOF:PRINT#3\\"No address stored":EXIT loop
160   READ name$,address$
170   IF name$==search$:PRINT #3\\name$,address$:EXIT loop
180 END REPeat loop
190 CLOSE #3
200 DATA 'Fred Blogs','17 Mulberry Court'
210 DATA 'John Peters','182 Johnson Ave.'
220 DATA 'Martin Edwards','83 Olive Drive'
```

```
100 OPEN_IN #3,flp1_Address_data
110 MODE 4 120 OPEN #4,con_448x200a32x16:BORDER#4,1,2:PAPER#4,0:INK#4,7
130 INPUT #4,'Input name to look for:'!search$
140 REPeat loop
150   IF EOF(#3):PRINT#4\\"No address stored":EXIT loop
160   INPUT #3;name$,address$
170   IF name$==search$:PRINT #4\\name$,address$:EXIT loop
180 END REPeat loop
190 CLOSE #4:CLOSE #3
```

**SMS NOTE**

Until v2.55 this command was the same as EOFW, which meant that it would only return a value if there was data waiting or it had received an end of file code - this was changed back to the original to maintain compatibility.

**CROSS-REFERENCE**

*DATA* specifies a line of data statements. *RESTORE* resets the data pointer and *READ* will actually fetch the data. *CLOSE* closes a given channel after it has been used. *PEND* or *IO_PEND%* are much better for use on pipes. See also *EOFW*.

## 12.26 EOFW

| | |
|---|---|
| Syntax | EOFW (#ch) |
| Location | SMS |

This function is very similar to EOF in that it returns the value 0 if there is data waiting to be read from the specified channel, otherwise it returns 1. The difference is that this version of the function will however

wait until data is received or the end of file code is received, which is especially useful on pipes which may not always work with EOF which returns 1 if the channel does not contain any data to be read.

**CROSS-REFERENCE**

See *EOF*. *PEND* and *IO_PEND%* are very similar.

## 12.27 EPROM_LOAD

| | |
|---|---|
| Syntax | EPROM_LOAD device_file |
| Location | ATARI_REXT (v1.21+), SMS |

You cannot plug QL EPROM cartridges into the various other computers which now support QL software, which would normally make some software which contains part of its code on EPROM, unusable. In order that you can use such software on other computers, you need to create a file on an original QL containing an image of the EPROM cartridge plugged into the QL's ROM port. To do this, use the command:

```
SBYTES flp1_EPROM_image,49152,16384
```

It is hoped that software producers who sell software which requires an EPROM cartridge will make versions available with ready-made images of the cartridge, so that the software can be used by users without access to an original QL.

Having done this, you will need to have the ST/QL Emulator switched on (or SMS loaded on the other computer), then insert that disk into the Atari's disk drive, and use the command: EPROM_LOAD flp1_EPROM_image This will then copy the EPROM code into the same address on the Emulator or other computer as the EPROM cartridge occupies on the QL, thus making it usable.

**NOTE 1**

If you make images of several EPROM cartridges in this way, then additional ones which are loaded with EPROM_LOAD will be stored in arbitrary addresses under SMS or the emulator. Therefore you will need to ensure that cartridges which insist on being loaded at the address $C000 (the QL's ROM port address), will need to be loaded first with EPROM_LOAD.

**NOTE 2**

On early versions of the Emulator, this was called ROM_LOAD.

**NOTE 3**

On SMS before v2.52, this could crash the system if used on a Gold Card or Super Gold Card without the specified file being present.

**CROSS-REFERENCE**

See also *ROM*, *ROMs* and ROM_TEST.

## 12.28 EPS

| | |
|---|---|
| Syntax | EPS [(x)] |
| Location | Math Package |

Since the precision of the QL is limited, a number may not change if a very small value is added. The function EPS(x) returns the smallest value which can be added to x so that the sum of x and EPS(x) will be different from x. This only makes sense for floating point numbers. The default parameter is 0. EPS(x) attains its smallest value at x=0, so EPS(0) returns the smallest absolute number which can be handled by SuperBASIC. EPS(x) is always greater than zero and EPS(x)=EPS(-x).

**Example**

An approximation of PI/4 as proposed by Leibniz:

```
100 x = 0: d = 1
110 t0 = DATE
120 FOR i=1 TO 1E100
130   IF ABS(1/d) < EPS(x) THEN EXIT i
140   x = x + 1/d
150   d = - SGN(d) \* (ABS(d)+2)
160 END FOR i
170 t = DATE - t0
180 PRINT "Iterations ="!i!" Runtime ="!t;"s"
190 PRINT "Iterations per Second ="!i/t
200 PRINT "PI ="!4\*x!"(";PI;")"
```

Unfortunately, the algorithm is not efficient enough to compete with the QL's precision, so that about 2E9 iterations are necessary to get a suitable result. Since this will take a while (ages!), you can reduce precision by a factor of one million, by modifying line 130:

```
130 IF ABS(1/d) < 1E6 * EPS(x) THEN EXIT i
```

The program will then finish after 1075 iterations with 4*x = 3.140662, not bad compared to 3.141593 when taking the drastic reduction of precision into account.

**NOTE**

EPS does not recognise the higher precision used by Minerva. Minerva's higher precision may have an effect on fractals and similar esoteric calculations.

## 12.29 EQ$

| | |
|---|---|
| Syntax | EQ$ (type, string1$, string2$ ) |
| Location | Btool |

This function expects the same parameters as GT$. It will return a value of 1 if the two strings are equal to each other using the same test as GT$.

**CROSS-REFERENCE**

See *GT$* for more details. NE is the same as:

```
NOT EQ$ (type, string1$,string2$)
```

## 12.30 ERLIN

| | |
|---|---|
| Syntax | ERLIN |
| Location | QL ROM (post JM version) |

This function returns the line where the last error occurred. If the error occurred in a line typed into the command window (#0), then zero is returned (zero is also returned if there is no error).

**Example**

It takes a lot of time to debug programs, so save typing by including a variation of the following line in your BOOT program. Then, if an error occurs and the program stops with an error message, simply press <ALT><E> to see and edit the line where something went wrong:

```
ALTKEY "e","ED ERLIN-20"&CODE(216)&CODE(216),""
```

or:

```
ALTKEY "e","AUTO ERLIN",""
```

**CROSS-REFERENCE**

*ERNUM* returns the error number, *REPORT* invokes an error message and *WHEN ERRor* allows error trapping. *ERLIN%* is exactly the same.

## 12.31 ERLIN%

| | |
|---|---|
| Syntax | ERLIN% |
| Location | Turbo Toolkit |

This function is exactly the same as ERLIN, except it will work on all versions of the QL ROM.

**CROSS-REFERENCE**

See *ERLIN* and *ERNUM%*.

## 12.32 ERNUM

| | |
|---|---|
| Syntax | ERNUM |
| Location | QL ROM (post JM version) |

This function returns the error number of the last error which occurred. An error number is negative and can be returned by any program (SuperBASIC, jobs, M/C Toolkits,...). The equivalent error messages are the same on all of the implementations of SuperBASIC, although they are also supported in different languages (see the Appendix for other languages):

| Error | English message |
|---|---|
| -1 | Not Complete |
| -2 | Invalid Job |
| -3 | Out of Memory |
| -4 | Out of Range |
| -5 | Buffer Full |
| -6 | Channel not Open |
| -7 | Not Found |
| -8 | Already Exists |
| -9 | In Use |
| -10 | End of File |
| -11 | Drive Full |
| -12 | Bad Name |
| -13 | Xmit Error |
| -14 | Format Failed |
| -15 | Bad Parameter |
| -16 | Bad or Changed Medium |
| -17 | Error in Expression |
| -18 | Overflow |
| -19 | Not Implemented Yet |
| -20 | Read Only |
| -21 | Bad Line |

**NOTE**

Jobs may return a positive error number. The meaning of such a number depends on the job. No error message will be reported.

**SMS NOTE**

The error messages have been redefined to try to make them more intelligent, they are now:

| Error | English message |
|---|---|
| -1 | Incomplete |
| -2 | Invalid Job ID |
| -3 | Insufficient memory |
| -4 | Value out of range |
| -5 | Buffer full |
| -6 | Invalid channel ID |
| -7 | Not found |
| -8 | Already exists |
| -9 | Is in use |
| -10 | End of file |
| -11 | Medium is full |
| -12 | Invalid name |
| -13 | Transmission error |
| -14 | Format failed |
| -15 | Invalid parameter |
| -16 | Medium check failed |
| -17 | Error in expression |
| -18 | Arithmetic overflow |
| -19 | Not implemented |
| -20 | Write protected |
| -21 | Invalid syntax |
| -22 | Unknown message |
| -23 | Access denied |

Other errors are reported by the SBASIC interpreter, but these are not covered by ERNUM.

**CROSS-REFERENCE**

*ERLIN* returns the line number where the error occurred. *ERNUM%* is the same as this function. *REPORT* invokes an error message and *WHEN ERRor* can be used to trap errors. The *ERR_XX* functions are alternatives to *ERNUM*.

## 12.33 ERNUM%

| | |
|---|---|
| Syntax | ERNUM% |
| Location | Turbo Toolkit |

This function is exactly the same as ERNUM, except it will work on all versions of the QL ROM.

**CROSS-REFERENCE**

See *ERNUM* and *ERLIN%*.

## 12.34 ERR_XX

| | |
|---|---|
| Syntax | ERR_NC, ERR_NJ, ERR_OM, ERR_OR, ERR_BO, ERR_NO, ERR_NF, ERR_EX, ERR_IU, ERR_EF, ERR_DF, ERR_BN, ERR_TE, ERR_FF, ERR_BP, ERR_FE, ERR_XP, ERR_OV, ERR_NI, ERR_RO, ERR_BL |
| Location | QL ROM |

These are logical functions which return either 0 or 1 if the corresponding error has occurred. Only one of them can have the value 1 at any time.

| Function | Error Code |
|---|---|
| ERR_NC | NOT COMPLETE -1 |
| ERR_NJ | INVALID JOB -2 |
| ERR_OM | OUT OF MEMORY -3 |
| ERR_OR | OUT OF RANGE -4 |
| ERR_BO | BUFFER OVERFLOW -5 |
| ERR_NO | CHANNEL NOT OPEN -6 |
| ERR_NF | NOT FOUND -7 |
| ERR_EX | ALREADY EXISTS -8 |
| ERR_IU | IN USE -9 |
| ERR_EF | END OF FILE -10 |
| ERR_DF | DRIVE FULL -11 |
| ERR_BN | BAD NAME -12 |
| ERR_TE | TRANSMISSION ERROR -13 |
| ERR_FF | FORMAT FAILED -14 |
| ERR_BP | BAD PARAMETER -15 |
| ERR_FE | FILE ERROR -16 |
| ERR_XP | ERROR IN EXPRESSION -17 |
| ERR_OV | ARITHMETIC OVERFLOW -18 |
| ERR_NI | NOT IMPLEMENTED -19 |
| ERR_RO | READ ONLY -20 |
| ERR_BL | BAD LINE -21 |

**NOTE 1**

These functions are not affected by REPORT.

**NOTE 2**

On Minerva pre v1.98, the ERR_XX functions were returning 1 if any higher error had occurred!!

**WARNING**

The JS ROM version of ERR_DF had a bug which crashed the system when used. All later operating systems and Toolkit II, the THOR XVI, the Amiga-QL Emulator, TinyToolkit, and BTool fix this.

**CROSS-REFERENCE**

See Appendix for other languages.

## 12.35 ERRor

| | |
|---|---|
| Syntax | ERRor |
| Location | QL ROM (post JM) |

This keyword forms part of the structure WHEN ERRor. Please refer to WHEN ERRor. As such, this keyword cannot be used in a program on its own - this will report 'bad line'.

**CROSS-REFERENCE**

*WHEN ERRor* contains a detailed description of this structure.

## 12.36 ERT

| | |
|---|---|
| Syntax | ERT function |
| Location | HOTKEY II |

Normally, whenever you use a function (or anything else which may return an error code), you will need to assign the result of the function (or whatever else) to a variable and then test that variable in order to see whether or not an error has been generated.

If an error has been generated, you will then need to report the error (if you do not intend to take any action to try and rectify the situation), something which can take a lot of program space, if you intend to write a program which does not require the command REPort to be present.

The command ERT was introduced in the Hotkey System II to enable you to write programs which test the result for an error code and report the error all in one step.

**Example 1**

A simple program which provides its own error trapping:

```
100 PAPER 0:INK 7
110 REPeat loop
120 CLS
130 AT 0,0:PRINT 'Enter an integer (0 to 300): ';
140 xerr=GET_INT
150 IF xerr<0:PRINT 'Error - try again':ELSE x=xerr:EXIT loop
160 PAUSE
170 END REPeat loop
180 PRINT 'The integer was : ';x
185 :
190 DEFine FuNction GET_INT
200   valid$='0123456789'
210   INPUT a$:IF a$='':RETurn -1
220   FOR i=1 TO LEN(a$):IF a$(i) INSTR valid$=0:RETurn -17
230   IF a$>300:RETurn -4
240   RETurn a$
250 END DEFine
```

**Example 2**

A similar program which is designed to stop on an error:

```
100 PAPER 0:INK 7
110 CLS
120 AT 0,0:PRINT 'Enter an integer (0 to 300): ';
130 xerr=GET_INT
140 IF xerr<0:REPORT xerr:STOP:ELSE x=xerr
150 PRINT 'The integer was : ';x
155 :
160 DEFine FuNction GET_INT
170   valid$='0123456789'
180   INPUT a$:IF a$='':RETurn -1
190   FOR i=1 TO LEN(a$):IF a$(i) INSTR valid$=0:RETurn -17
200   IF a$>300:RETurn -4
210   RETurn a$
220 END DEFine
```

**Example 3**

The same program as in the second example, but using ERT:

```
100 PAPER 0:INK 7
110 CLS
120 AT 0,0:PRINT 'Enter an integer (0 to 300): ';
130 ERT GET_INT
140 PRINT 'The integer was : ';x
150 DEFine FuNction GET_INT
160   valid$='0123456789'
170   INPUT a$:IF a$='':RETurn -1
180   FOR i=1 TO LEN(a$):IF a$(i) INSTR valid$=0:RETurn -17
190   IF a$>300:RETurn -4
200   x=a$
210   RETurn x
220 END DEFine
```

**NOTE**

When you are using ERT, always beware of what you are testing for an error, for example, if you had altered line 130 in the second example to:

```
130 ERT x=GET_INT
```

you would not actually be testing to see whether the function GET_INT returned an error, but whether the line x=GET_INT produced an error - x itself would not be altered, hence the need to assign the result to x inside the function.

**CROSS-REFERENCE**

*REPORT* will report an error without stopping the program.

## 12.37 ESC

| | |
|---|---|
| Syntax | ESC |
| Location | Beuletools |

This function returns the control codes ESC, or CHR$(27) for use on an EPSON compatible printer:

```
PRINT ESC
```

is the same as:

```
PRINT CHR$(27)
```

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*,*DBL*,*ENL*,*PRO*,*SI*,*UNL*,*ALT*,*FF*,*LMAR*,*RMAR*,*PAGDIS*, *PAGLEN*. *UPUT*

## 12.38 ET

| | |
|---|---|
| Syntax | ET file *[,{filex \| #chx}]* [;cmd$] |
| Location | Toolkit II |

The syntax for ET is the same as for the Toolkit II variant of EX and it also operates in a similar manner. However, ET is intended for low level debugging, ie. to trace execution of the machine code commands step by step.

A monitor program such as Qmon is necessary.

The command ET loads the executable program, installs the job and immediately suspends the job by setting its priority to zero. Control is then returned to SuperBASIC to allow you to use a monitor program.

**CROSS-REFERENCE**

*EX*

## 12.39 ETAB$

| | |
|---|---|
| Syntax | ETAB$ (string$ [,tabdist]) where tabdist=1..255 |
| Location | BTool |

Some editors and word-processors use the character CHR$(9) as a tab mark to save the space which would otherwise be needed to store several spaces. The function ETAB$ takes a given string, expands all tab marks in it and returns the result.

If the tabulator distance, tabdist, is not given, a default of eight characters is assumed. The length of string$ has to be smaller than 256 characters: LEN(string$)<256.

Tabdist>255 has no effect.

**Example**

The text file test_txt is shown with all tab marks expanded:

```
100 OPEN_IN#3,test_txt
110 CLS
120 REPeat all_lines
130   IF EOF(#3) THEN EXIT all_lines
140   INPUT#3,line$
150   IF LEN(line$)>255 THEN line$=line$(1 TO 255)
160   PRINT ETAB$(line$,4)
170 END REPeat all_lines
180 CLOSE#3
```

**NOTE**

A value of tabdist<=0 will not produce usable output.

**WARNING**

Although tab mark distances of 32766 and 32767 are allowed, ETAB$ will not produce a sensible output. It may even possibly crash the system.

**CROSS-REFERENCE**

*CTAB$* is the complimentary function to *ETAB$*. *INSTR* finds the position of a string in another string. *LEN* returns the length of a string.

## 12.40 ETAT

| | |
|---|---|
| Syntax | ETAT (file$) |
| Location | ETAT |

This function checks to see if the given file (passed as a string) exists and then checks upon its status (whether it can be opened etc). If necessary a standard error number is returned, otherwise ETAT will return 0, which means that the file can be accessed without the danger of an error such as "not found". This can therefore be used to avoid the need for error trapping.

**Example**

This program copies text files to window #1:

```
100 REPeat input_loop
110 INPUT "File to view:"!file$
120   AnError=ETAT(file$)
130   IF NOT AnError: EXIT input_loop
140   PRINT "Sorry, ";: REPORT#1,AnError
150 END REPeat input_loop
160 OPEN_IN#3,file$
170 REPeat view_file
180 IF EOF(#3) THEN EXIT view_file
190 INPUT#3,line$: PRINT line$
200 END REPeat view_file
210 CLOSE#3
```

**CROSS-REFERENCE**

*FTEST* works like *ETAT* but recognises the default device and directory. *FILE_OPEN*, *FOPEN*, *FOP_IN*, *FOP_OVER* and *FOP_NEW* are all functions to open files without the need for error trapping. *OPEN*, *OPEN_IN* and *OPEN_NEW* stop with error messages if an error occurs. To avoid this, error trapping facilities, such as *WHEN ERRor* have to be used.

## 12.41 EW

| | |
|---|---|
| Syntax | EW file *[,{file$^x$ \| #ch$^x$}]* |
| Location | Toolkit II, THOR XVI |

This command causes the given file (which must be an executable program) to be executed.

If the drivename is not given, or the file cannot be found on the given device, EW will load the first file from the default program directory (see PROGD$), with subsequent programs being loaded from the default data directory (see DATAD$). The calling program will be stopped whilst the new job is running (ie. the new job cannot multitask with the calling program). If you supply any channels (which must already be open in the calling program) or filenames as parameters, these form channels which can be accessed by the job.

If your program has been compiled with QLiberator or is to be run as an SBASIC job under SMS then each supplied channel will become #0, #1, #2 ....

Note that with Turbo compiled programs the channels work backwards and will become #15, #14, #13 ... To access these channels from within the job, merely ensure that the job does not try to open its own channel with the same number, and then write the program lines as if the channels were open. Further, you can pass a command string (cmd$) to the program specifying what the executed job should do. It depends on the job what cmd$ should look like and also how you will access the given string. The Turbo and QLiberator compilers include commands in their Toolkits to read the supplied string; and Minerva MultiBASICs and SMS SBASICs include the function CMD$ which allows you to read the supplied string.

If you have not used one of these compilers to produce the job, then you will need to read the string from the stack. Please note that the command string must appear as the last parameter for the command. The command string can be explicit strings and names as well as expressions. However, variables must be converted into expressions, for example by:

```
EW 'flp1_xchange';(dataspace)
```

On some very early versions of Toolkit II, you needed:

```
EW 'flp1_xchange';dataspace&""
```

Executable programs often return an error code back to the owner job (the program which started it). Especially with 'C' compiled programs, this will be non-zero if there are any errors. EW stops the owner job if this happened. There is unfortunately no way to stop this from happening unless you use error trapping (eg. WHEN ERRor, or Q_ERR_ON from QLiberator).

**Example 1**

```
EW QED;"flp1_readme_txt"
```

The editor will be started from the default program directory and told to load the file readme_txt.

**Example 2**

```
EW mdv1_QUILL
```

will start QUILL from microdrive 1.

**NOTE 1**

There are problems with EW and EX in Toolkit II v2.05 (and previous versions) which make them unreliable and difficult to use with compiled programs. The main problem lay in what was classed to be the owner of a secondary Job. From v2.06 onwards, the owner for EX has been Job 0 and the owner for EW, the current Job.

**NOTE 2**

TinyToolkit and BTool allow you to break out of a program started with EW at any time by pressing <CTRL><SPACE> - the program can then be treated as if it was started with EX.

**NOTE 3**

On some versions of the QL ROM (and Toolkit II), unless the Pointer Environment is loaded, you may need to press <CTRL><C> to get back the cursor at the end of the task.

**NOTE 4**

You cannot use EW (or similar) to execute a file stored on a PC or TOS disk (even with Level-3 Device Drivers) - see the Device Drivers Appendix, in particular the notes on Level-3 Device Drivers for further details.

**MINERVA NOTES**

As from v1.93+, MultiBASICs can be started up with the command:

EW pipep $^{*}$[,{file$^{x}$ | #ch$^{x}$}]$^{*}$ [;cmd$]

Prior to this version, you needed to load the file Multib_exe contained on the disk supplied with Minerva and use the command:

EW flp1_Multib_exe $^{*}$[,{file$^{x}$ | #ch$^{x}$}]$^{*}$ [;cmd$]

How any supplied channels are dealt with is slightly different to all other implementations. Its effect depends on how many channels are passed:

- No channels passed - MultiBASIC started with a single small window which is the same for #0 and #1.
- One channel passed - This becomes both #0 and #1.
- Two channels passed - These become #0 and #1 respectively.
- Three or more channels passed - The first two become #0 and #1 respectively, then any additional ones become #3 onwards.

Minerva MultiBASICs also treat any command string passed to them in a special way:

- If the last character of the command string is an exclamation mark (!), then the MultiBASIC is started up with the original keywords built into the ROM, and any which had been linked into SuperBASIC subsequently (for example Toolkit II) will not be available to that MultiBASIC. This character is then removed from the command string before it can be read by the MultiBASIC.

- If the command string contains the greater than sign (>), then anything which appears before that character in the string, is opened as an input command channel (thus allowing you to run a Multi-BASIC program in the background) and then all characters up to and including the greater than character are deleted from the command string before it can be read by the MultiBASIC.

**Example**

Take a simple BASIC program to convert a given file (say flp1_TEST_TXT) into uppercase:

```
110 REPeat loop
120   IF EOF(#0) THEN EXIT loop
130   INPUT #0,a$
140   IF a$='' THEN NEXT loop
150   FOR i=1 TO LEN(a$)
160     IF CODE(a$(i))>96 AND CODE(a$(i))<123 THEN
170       a$(i)=CHR$(CODE(a$(i))-32)
180     END IF
190   END FOR i
200   PRINT a$
210 END REPeat loop
220 IF VER$(-1):CLOSE #0
```

Save this as flp1_UC_bas and then enter the command:

```
OPEN #3,con
EW pipep,flp1_test_txt,#3;'flp1_UC_bas>'
```

or, prior to v1.93, use:

```
OPEN #3,con
EW flp1_Multib_exe,flp1_test_txt,#3;'flp1_UC_bas>'
```

The last line checks to make sure this program is not being run from the original SuperBASIC interpreter (job 0) in which case, it then closes #0. Unfortunately, on v1.97 (at least), this program fails to spot the end of the file (try PEND instead of EOF), and therefore reports an 'End of File' error on completion. Oddly, this error is not reported if you use EX to run the program.

**SMS NOTE**

SMS allows EW and EX to run basic programs in the background, as an SBASIC job. For example, using the Minerva example program above, this could be used with the line:

```
EW flp1_UC_bas,flp1_test_txt,#3
```

This does not report an error on completion. Beware however that prior to v2.69, this command would not work in Qliberated programs to start an SBASIC program. Because of this ability, SMS v2.58+ has amended the EW set of commands so that it searches for a file in much the same way as LOAD under SMS.

Taking a default program device to be flp1_,

```
EW ram1_TEST
```

will look for the following files:

- ram1_TEST

- ram1_TEST_sav
- ram1_TEST_bas
- flp1_ram1_TEST
- flp1_ram1_TEST_sav
- flp1_ram1_TEST_bas

**CROSS-REFERENCE**

For further information see *EX*. *SBASIC* allows you to set up several SBASIC jobs under SMS. *MB* allowed you to start up a MultiBASIC on early versions of Minerva. Please also see the appendix on Multiple BASICs.

## 12.42 EX

| | |
|---|---|
| Syntax | EX file *[,{file$^x$ \| #ch$^x$}]* [;cmd$] |
| Location | Toolkit II, THOR XVI |

This command forces the given file (which must be an executable program) to be executed and control is then generally returned to the calling program to enable the new job to multitask alongside the calling program. Similar parameters as for *EW* can be passed to the job.

Use *EW* if the program cannot multitask for some reason or if you do not want it to, for example, because you want to see any error messages returned by the executable task. EX doesn't report them, it cannot as the executable task may still be running when EX returns to the command prompt.

**Example 1**

```
EX QED;"readme_txt"
```

The QED editor will be started from the default program device and told to load the file readme_txt from the editor's default device.

**Example 2**

```
EX UC_obj,ram1_hope_lis,par
```

A program called UC_obj (a program which converts text to all upper case) will be started up to run alongside all other programs. Two n:ref:*ew* channels ('ram1_hope_lis' and 'par') are opened for the task to use for its input and output channels respectively - the task must not open its own channels but will rely upon the user supplying them as parameters.

The BASIC version of such a program is:

```
110 REPeat loop
120 IF EOF(#0) THEN EXIT loop
130 INPUT #0,a$
140 IF a$='' THEN NEXT loop
150 FOR i=1 TO LEN(a$)
160   IF CODE(a$(i))>96 AND CODE(a$(i))<123 THEN
170     a$(i)=CHR$(CODE(a$(i))-32)
```

(continues on next page)

(continued from previous page)

```
180   END IF
190 END FOR i
200 PRINT#1,a$
210 END REPeat loop
```

Turbo users will need to alter #0 and #1 to #15 and #14 respectively.

Minerva and SMS users can use this program without compiling it (see *EW* above).

**Using EX to set up filters**

It is actually quite simple to create a multitasking environment on the QL using the EX command to set up several programs all of which will process a given file (or data entered into a given channel) in turn.

The syntax for this version of the command is:

EX jobparams$^{1*}$[TO jobparams$^{i}$]$^{*}$ [TO #chan$^{0}$]

where jobparams represents the same parameters as are available for the normal EX command, being:

file $^{*}$[,{file$^{x}$ |#ch$^{x}$}]$^{*}$ [;cmd$]

What this actually does, is to set up a chain of jobs or channels whereby one extra channel is opened for each job to form the output channel for the job on the left of the TO and another channel is opened to form the input channel of the job on the right of the TO.

Where a channel number appears at the end of the line (after a TO), this is taken as being the final output channel and nothing further can be done to the original input.

**Examples**

How about extending the Upper case conversion 'filter' so that a given text file is then printed out one line at a time with each line being printed out as normal, but then printed again, but this time backwards!

First of all, the program to do the printing:

```
110 REPeat loop
120   IF EOF(#0): EXIT loop: REMark Turbo uses #15, not #0
130   INPUT #0,a$:PRINT#1,a$: REMark Turbo uses #14, not #1
140   IF CMD$=='y': REMark Turbo users use OPTION_CMD$
150   IF a$='':NEXT loop
160   FOR lop=LEN(a$) TO 1 STEP -1
170     PRINT#1,a$(lop);
180   END FOR lop
190   PRINT#1
200   END IF
210 END REPeat loop
```

Compile this program and save the compiled version as flp1_Back_obj.

Now to carry out the desired task:

```
OPEN #3,con
EX flp1_uc_obj,flp1_test_txt TO flp1_back_obj,#3;'y'
```

On Minerva v1.93+, you could use:

```
OPEN #3,con
EX pipep,flp1_test_txt;'flp1_uc_bas>' TO pipep,#3;'flp1_back_bas>y'
```

Or on SMS:

```
OPEN #3,con
EX flp1_uc_bas,flp1_test_txt TO flp1_back_bas,#3;'y'
```

How about trying this:

```
OPEN #3,con
EX flp1_uc_obj,flp1_test_txt TO flp1_back_obj;'y' TO flp1_back_obj,#3;'y'
```

**NOTE 1**

On pre JS ROMs, you may find that if you EX a new Job whilst there is already one Job in progress, the ink and paper colours of the first Job are set to zero. This is a problem unless you have a key to redraw the screen for the first Job (or the Pointer Interface).

**NOTE 2**

The THOR XVI always ensures that cursor control is passed to the new Job on start-up rather than returning to the calling Job.

**MINERVA NOTE**

Please refer to notes about *EW* which explain how to use this command to control MultiBASICs.

**SMS NOTE**

Please refer to notes about *EW* and use this command to control multiple SBASICs.

**CROSS-REFERENCE**

Use *FTYP* or *FILE_TYPE* to check if a file is executable. *FDAT* returns the dataspace of an executable file, *FXTRA* provides other information. *ET* is very similar to *EX*.

## 12.43 EXCHG

| | |
|---|---|
| Syntax | EXCHG device_file,old$,new$ |
| Location | ATARI_REXT |

This command creates a Job which opens a channel to the specified file and then works through the file, replacing every occurrence of old$ with new$. The search for old$ is case independent. Both old$ and new$ must be the same length.

**Example**

```
EXCHG flp1_Task_obj,'mdv','flp'
```

will replace all references to mdv1_ or mdv2_ to flp1_ and flp2_ respectively in the file flp1_task_obj.

**NOTE**

CHR$(0) cannot be replaced!

**CROSS-REFERENCE**

See also *CONVERT*.

## 12.44 EXEC

| | |
|---|---|
| Syntax | EXEC program or<br>EXEC file *[,{file$^x$ \| #ch$^x$}]* [;cmd$] (Toolkit II, THOR XVI) or<br>EXEC file *[,#ch$^x$]* [;cmd$] (Minerva v1.93+) |
| Location | QL ROM, Toolkit II |

This command loads and starts a machine code or compiled program, but then returns control to the calling job (ie. the job which issued EXEC) so that both jobs are multitasking.

Minerva v1.97+ has now implemented a sub-set of the Toolkit II standard, in that you can pass details of existing channels to a job as well as a command string.

**CROSS-REFERENCE**

With Toolkit II installed or on a THOR XVI, *EXEC* is the same as *EX*. See also *EXEC_W*, *EW*, *TTEX* and *ET*. If you are using the Hotkey System or SMS then see *EXEP* in this manual.

## 12.45 EXEC_W

| | |
|---|---|
| Syntax | EXEC_W program or<br>EXEC_W file *[,{file$^x$ \| #ch$^x$}]* [;cmd$] (Toolkit II, THOR XVI) or<br>EXEC_W file *[,#ch$^x$]* [;cmd$] (Minerva v1.93+) |
| Location | QL ROM, Toolkit II |

This command is the same as EXEC except that the calling job is suspended until the program has finished.

**CROSS-REFERENCE**

Toolkit II and a THOR XVI make *EXEC_W* the same as *EW*. See also *EXEC*, *EX*, *TTEX*, *TTEX_W* and *ET*.

## 12.46 EXEP

| | |
|---|---|
| Syntax | EXEP filename [;cmd$] [,Jobname$] [,options] or<br>EXEP Thingname$ [;cmd$] [,Jobname$] [,options] (version 2.17+) |
| Location | HOTKEY II |

The first variant of the EXEP command is similar to the EX and EW commands provided by Toolkit II. However, not only does EXEP allow you to pass a command string to the program being called (as with

EX or EW), but you can also supply the Job name which will be shown in a list of the Jobs currently loaded into memory.

In order to make various 'problem' programs work correctly under the Pointer Environment, it is sometimes necessary to pass various parameters (options) to the Hotkey System when the program is called in order to tell it how to treat the program.

The command EXEP allows you to execute a program (in the same way as with EXEC), but at the same time, pass these parameters to the Pointer Environment. The parameters (or options) currently supported are:

- P [,size]- This tells the Hotkey System that the program is a Psion program (eg. Quill) which will try to grab all of the available memory.

  If size is not specified, then the Hotkey System will ask the user to specify the maximum amount of memory (in kilobytes) that the program should use before the program actually starts. Otherwise, the program will be allowed to use size kilobytes of memory (if available).

  When the Pointer Environment was first released, Qjump produced a program (Grabber) which could be used to amend the amount of memory addressed by the Psion programs once and for all - if this program has been used on your copies of the Psion programs, then do not use this option.

- G [,x,y,a,b] - When a program is started, the Pointer Interface will store the area of the screen contained under each window as it is opened, restoring any part of the screen is no longer covered by an active window.

  This provides non-destructive windows, one of the major assets of the Pointer Interface. However, some programs have a habit of opening windows, writing to the screen and then closing the window so that the text cannot be altered - creating background information.

  Unfortunately, due to the way in which the Pointer Interface works, as soon as this window is closed, the background information would be lost.

  The solution to this is to use a guardian window (created using this parameter) which specifies the area of the screen which the program is allowed to use and which must therefore not be restored until the program has ended (even if there are no current windows open on that area). The parameters are used to open a guardian window x pixels wide by y pixels high at the origin (a,b).

  Any attempt by a program to open or resize a window so that part of it would fall outside this Guardian window will fail.

  If you do not pass the size of the Guardian window as a parameter (eg. EXEP flp1_Graph_exe,g), the maximum permissible window size will be assumed (eg. 512x256 on a standard QL).

- F - Some programs which use KEYROW to read the keyboard, or access the screen directly, can wreak havoc when multitasking alongside other programs.

  This parameter causes the computer to only pass any keypresses read with KEYROW to the program started with EXEP.

- U - With some programs, for example, a clock, it is desirable for this to be updated on screen even though it is not the Job at the top of the pile (ie. it is overwriting part of the current Job's windows).

  The Pointer Interface will allow you to do this by passing the u parameter (for unlock), for example:

  ```
  EXEP flp1_Clock,u
  ```

The second syntax of EXEP is similar, except that instead of loading a task stored with the given filename, it searches through the Thing list for an Executable Thing with the given Thingname and then (if present),

will start that up as a new Job (if it is not present, then EXEP will look on the default program device for a file called Thingname).

For example, if you have QPAC2 present, EXEP Files will call up the files sub-menu (in the latest versions of QPAC2, you could use, for example:

```
EXEP files;'\S \D flp1__exe \O v','View _EXE'
```

to create a View files menu which will list all of the files on flp1_ which end with _exe, without any sort order; the job being called 'View _EXE' in the Jobs list).

**Example 1**

Consider the following program:

```
100 MODE 4
110 OPEN #0,CON_10x10a132x66
120 OPEN #1,CON_448x200a32x16
130 PAPER 0:INK 7:CLS
140 BORDER 1,2:AT 10,9:PRINT 'Y AXIS'
150 AT 15,35:PRINT 'X AXIS'
160 OPEN #1,CON_248x100a132x66:BORDER 1,4
170 PAUSE
```

If this program was compiled (without windows being copied across) and then run, as soon as line 160 was reached, the information around the sides of the graph would be lost! The reason for the PAUSE in line 170 is that as soon as the compiled program reached the end, it would close all of its windows, and you would not be able to see anything! The answer is to use a Guardian window (created using this parameter). Presuming that the above program has been compiled under the filename flp1_Graph_exe, you could use the line:

```
EXEP flp1_Graph_exe,G,448,200,32,16
```

to define a Guardian window 448x200 pixels with its origin at (32,16).

**Example 2**

Try for example, compiling the following program and starting it with:

```
EXEP flp1_Test_exe,u
```

(presuming that is the filename you allocate to it):

```
100 OPEN #1,con_512x256a0x0
110 REPeat Loop
120   PRINT KEYROW(0)
130 END REPeat Loop
```

You will find it very difficult to do anything (including removing this job). The solution is to pass this parameter to the Pointer Interface which tells it to Freeze the program when it is in buried under another Job's windows (eg. if you used <CTRL><C> to change to another Job). For example, use the line:

```
EXEP flp1_Test_exe,f
```

**Example 3**

The SuperBASIC line:

```
EXEP flp1_EDT;'flp2_Text',Editor,g
```

will start up an editor stored under the filename flp1_EDT, which will be given the Job name 'Editor' (which will be shown for example in the JOBS table), provide it with a guardian window of 512x256, and tell it to load a file called flp2_Text.

**NOTE 1**

Before v2.21 of the Hotkey System II, you could not pass a command string to the program being called.

**NOTE 2**

The various parameters can be mixed together, for example:

```
EXEP flp1_Graph_exe,F,G,448,200,32,16;'ser1'
```

**NOTE 3**

Versions earlier than v2.24 will not allow you to alter the Job Name, which will otherwise be the name given the program when it was created.

**CROSS-REFERENCE**

*THING* allows you to test whether or not a given Thing is present. *EX*, *EXEC*, *EW* and *EXEC_W* are all similar to the first variant of *EXEP*. *GET_STUFF$* will call up the QPAC2 files sub-menu and allow you to read the chosen filename. *HOT_THING* allows you to set up a hotkey to call an Executable Thing.

## 12.47 EXIT

| | |
|---|---|
| Syntax | EXIT loop_variable (FOR loops) or<br>EXIT loop_name (REPeat loops) or<br>EXIT(SMS only) |
| Location | QL ROM |

Using the first two variants of this command, the specified loop (either a FOR or a REPeat structure) will be finished and the program will jump to the first statement after the relative END FOR loop_variable or END REPeat loop_name.

The third variant only exists under SMS and will force the interpreter to jump out of the current loop being executed, whether it is a FOR loop or a REPeat loop - the interpreter will just search the program for the next END REPeat or END FOR statement.

**NOTE 1**

If two or more loops are nested together, it is possible to EXIT the outer loop from within the inner loop:

```
REPeat loop1
  ...
  REPeat loop2
    ...
    IF condition THEN EXIT loop1 ---+
```

(continues on next page)

(continued from previous page)

```
    ...                                |
  END REPeat loop2                     |
  ...                                  |
END REPeat loop1                       |
...                       <------------+
```

Such a structure is not regarded as elegant by some people because it is not possible to draw a structogram from this.

**NOTE 2**

If a program is badly written, this can lead to confusion - for example, try:

```
100 REPeat loop
120   PRINT 'Hello'
130   EXIT loop
140 END REPeat loop
150 END REPeat loop
```

The interpreter fails to notice the misplaced END REPeat at line 150.

The first time that EXIT loop is encountered, the interpreter leaves the loop at line 140 - however, line 150 forces the interpreter to execute the loop a second time. This time, EXIT loop forces the interpreter to jump out the loop at line 150. The same thing happens if you use FOR … END FOR instead of REPeat … END REPeat

This feature allows you to jump back into a loop from anywhere in the program (although this should be avoided). Compare what happens if NEXT loop is used instead of END REPeat loop in line 150, EXIT loop will always exit the loop at line 140. This means that NEXT loop can also be used to jump back into a loop from anywhere in the program (although again, this should be avoided).

Note that in any event, these latter two features will only work if the named loop has already been RUN (setting up the loop variables)!!

**CROSS-REFERENCE**

Please see *FOR* and *REPeat* for more details.

## 12.48 EXP

| | |
|---|---|
| Syntax | EXP (var) |
| Location | QL ROM |

This function returns the value of the mathematical base e to the power of the given parameter (in other words, this is equivalent to the mathematical expression $e^{var}$). This is the opposite to the function LN, ie. var=LN(EXP(var)).

QDOS supports var in the range -512…511. The approximate value of e can be found by:

```
PRINT EXP(1)
```

```
PRINT EXP(0)
```

returns the value 1 - as any good mathematician knows, anything to the power of 0 returns the value 1.

**CROSS-REFERENCE**

*LN* returns the natural logarithm of the given value.

## 12.49 EXPAND

| | |
|---|---|
| Syntax | EXPAND file$ |
| Location | COMPICT |

This command takes a screen file (which must have been created with COMPRESS), and re-expands it on the visible screen.

**NOTE 1**

EXPAND needs a working space of 32K. A memory overflow error will be reported if there is not enough memory available.

**NOTE 2**

EXPAND assumes that the screen starts at $20000 and will therefore not work on Minerva's second screen or extended resolutions.

**NOTE 3**

EXPAND will not work on QLs with resolutions above 512x256

**WARNING**

If the file was not saved by COMPRESS, it is most likely that the system will crash. This is certain if the file is longer than 32K.

**CROSS-REFERENCE**

*COMPRESS*, *FASTEXPAND*.

## 12.50 EXPLODE

| | |
|---|---|
| Syntax | EXPLODE |
| Location | ST/QL, QSound |

This command produces the sound of an explosion, very nice.

**CROSS-REFERENCE**

*SND_EXT*, *BELL*, *SHOOT*.

## 12.51 EXTRAS

| | |
|---|---|
| Syntax | EXTRAS [#channel] or<br>EXTRAS \file (Toolkit II, THOR only) or<br>EXTRAS [#channel][,width] (BTool only) |
| Location | Toolkit II, THOR XVI, QSound, BTool |

This command lists all of the machine code Procedures and Functions (keywords) which are recognised by the SuperBASIC interpreter in the given channel (default #1), or the given file (if the second variant is used), which will be automatically opened and even overwritten if it already exists (after asking the user to confirm that this is okay).

The file will be closed at the end of the operation.

The THOR XVI version will not list those keywords which are resident in ROM (ie. available when the THOR is first powered up).

The BTool version lists the keywords in columns and as such is the same as EXTRAS_W. The number of columns is adapted automatically to a window's width; if this is too wide for your needs then you can specify a width in characters.

The QSound variant is intended for output to a non-screen channel (see WIDTH), in which case an empty line appears between each name. If output is sent to a window, then the words are all printed on the same line, obscuring output.

**NOTE 1**

BTool's EXTRAS does not support the SuperBASIC WIDTH command and you will therefore need to specify an absolute width as the second parameter to format output.

**NOTE 2**

Versions of Tiny Toolkit pre v1.10 contained a different implementation of this command, now renamed TXTRAS.

**NOTE 3**

Within an SBASIC (on SMS), EXTRAS only lists those keywords used in that SBASIC to date - this is because the whole name table is not copied when an SBASIC is started up, allowing different SBASICs to use the same name for different things.

**CROSS-REFERENCE**

Use *SXTRAS* if you have a lot of extensions in memory and you are looking for a specific one. See also *TXTRAS*, *VOCAB* and *NEW_NAME*.

## 12.52 EXTRAS_W

| | |
|---|---|
| Syntax | EXTRAS_W [#ch] |
| Location | ATARI_REXT |

This lists all of the current SuperBASIC commands to the given channel (default #1). Unlike EXTRAS, the output appears in columns and there is no pause when the given window is full.

### CROSS-REFERENCE

*EXTRAS* is very similar.

CHAPTER

THIRTEEN

# KEYWORDS F

## 13.1 FACT

| | |
|---|---|
| Syntax | FACT(n) where n=0..300 |
| Location | Math Package, FACT |

The FACT function takes a non-negative integer n up to 300 and returns the factorial of the number, calculated as the product: 1*2*3*…*n

**Example**

n elements can be combined in FACT(n) different ways, eg. take the three first letters, the FACT(3)=6 permutations of A, B and C are:

1. ABC
2. ACB
3. BAC
4. BCA
5. CAB
6. CBA

**CROSS-REFERENCE**

*BINOM*

## 13.2 FALSE%

| | |
|---|---|
| Syntax | FALSE% |
| Location | TRUFA |

The function FALSE% returns the constant 0. It is used to write programs which are more legible or which adopt habits from the PASCAL language.

**CROSS-REFERENCE**

*TRUE%*. See also *SET* concerning user definable resident constants.

## 13.3 FASTEXPAND

| | |
|---|---|
| Syntax | FASTEXPAND adr1,adr2 |
| Location | COMPICT |

If a screen which has been compressed and saved with COMPRESS is loaded into memory with LBYTES (for example), this command allows quicker expansion of the screen than would otherwise be possible with EXPAND.

FASTEXPAND also allows you to expand the screen to any address (provided that there is at least 32K of free memory stored there). adr1 is the address where the compressed screen is stored and adr2 the place in RAM where the expanded screen should be moved to.

**Example**

```
100 COMPRESS ram1_test_scr
110 a=ALCHP(FLEN(\ram1_test_scr))
120 LBYTES ram1_test_scr,a
130 FASTEXPAND a,SCREEN
140 RECHP a
```

**NOTE**

FASTEXPAND will not work on screen resolutions in excess of 512x256 pixels.

**CROSS-REFERENCE**

*COMPRESS*, *EXPAND*.

## 13.4 FAT_DRIVE

| | |
|---|---|
| Syntax | FAT_DRIVE drive, card, partition |
| Location | SMSQ/E for Q68 |

This command mounts a the primary FAT16 partition (1-4) on an SD card (1-2) as FAT_drive to simplify moving software into the Q68.

**Example**

```
FAT 1,1,2
```

**Note 1**

Do not attempt to make two FAT drives point to the same partition!

**Note 2**

The partition must be a primary partition, formatted FAT16, and no larger than 255 MB.

**Note 3**

Normally partition 1 is formatted FAT32 to allow it to contain WIN images.

**Note 4**

By default the FAT driver may not be loaded in SMSQ/E

**CROSS-REFERENCE**

*FAT_DRIVE$* returns the parameters already associated with a FAT drive.

## 13.5 FAT_DRIVE$

| | |
|---|---|
| Syntax | FAT_DRIVE$ (drive) |
| Location | SMSQ/E for Q68 |

This function returns a string containing the card and partition for specified FAT drive.

If the specified drive has not been linked to any particular hard disk partition, an empty string is returned.

**Example**

```
FAT_DRIVE 1,1,2
PRINT FAT_DRIVE$(1): REMark Will print Card: 1, Partition: 2
```

**Note 1**

By default the FAT driver may not be loaded in SMSQ/E

**CROSS-REFERENCE**

See *FAT_DRIVE*.

## 13.6 FAT_USE

| | |
|---|---|
| Syntax | FAT_USE [device] |
| Location | SMSQ/E for Q68 |

As with FLP_USE this allows you to assign another three letter description to the FAT device driver, so that it can be accessed by programs which do not allow you to alter their devices. If no device is specified, then the device name is returned to the default fat.

**Example**

```
FAT_USE mdv
```

will ensure that any further attempt to access mdv1_ will actually access fat1_. If you later use the command:

```
FAT_USE
```

or:

```
FAT_USE fat
```

then you will once again be able to use the microdrives as well as fat1_.

**Note 1**

The QL's operating system tests for directory device drivers in a fixed order: DEV, FLP, RAM, WIN and MDV. This means that if you rename a driver to three letters which refer to a device driver earlier in the list, that original device driver will be used in preference. For example:

```
FAT_USE flp
```

will not work (attempts to read a file from flp1_ will still try to read floppy disk drive number one) - you will need to also rename the floppy disk driver:

```
FLP_USE flp
```

**Note 2**

By default the FAT driver may not be loaded in SMSQ/E

**CROSS-REFERENCE**

*FLP_USE*, *RAM_USE*, *DEV_USE*, *WIN_USE* are similar. *DMEDIUM_TYPE* can be used to find out the type of device which a name actually refers to. *DMEDIUM_NAME$* will return the default name of a device.

## 13.7 FAT_WP

| | |
|---|---|
| Syntax | FAT_WP drive, protect |
| Location | SMSQ/E for Q68 |

This command allows you to mark a specified FAT drive as read only protect=1 will write protect the hard disk. protect=0 (the default) will remove the write protection.

**Note 1**

By default the FAT driver may not be loaded in SMSQ/E

**CROSS-REFERENCE**

*DMEDIUM_RDONLY* will tell you if a device is read only.

## 13.8 FBKDT

| | |
|---|---|
| Syntax | FBKDT [(#channel)] or<br>FBKDT (\file) |
| Location | Level-2 Device Drivers, SMS |

It is proposed that this function be used to return the date on which a given file was last backed up. Current versions of SuperBASIC commands, such as COPY and WCOPY do not set the back-up date of the file being copied, although some software will do this, WinBack for example.

The idea of a back-up date is mainly for use in automatic back-up programs which can be told to copy all files on a given medium within certain parameters, namely files which have been altered since a specific date and which have been altered since the last time that they were backed up.

The value returned is the date in QDOS format, ie. the number of seconds since Midnight 1st January 1961 {check this initial date with PRINT DATE$(0)}. This backup time currently needs to be set manually using SET_FBKDT, although it is hoped that future versions of COPY and WCOPY will do so automatically.

If it has not been set, FBKDT will return zero. The default data device and sub-directories are supported, default channel is #3.

**Example**

The PROCedure below will make an intelligent backup of all files contained in the medium specified by the first parameter to the medium specified in the second parameter, which have been altered since they were last backed up. TinyToolkit's *TCONNECT* or DIY-TK's *QLINK* is necessary to use this example. This can be used for example by entering the line:

```
BACKUP flp1_ TO flp2_
```

Although sub-directories and the default data device are fully supported on the medium being backed-up, the procedures would need modification to enable them to create similar sub-directories on the destination device. The PROCedure makes heavy use of recursive programming, which means that it uses a lot of memory (not all of which is released at the end of the PROCedure). It would need a considerable re-write to be in a form which could be safely compiled.

```
100 DEFine PROCedure BACKUP (dir1,dir2)
110   LOCal dir1$,dir2$,old_datad$,old_destd$
120   LOCal ERRno,outer,sloop
130   dir1$=PARSTR$(dir1,1):dir2$=PARSTR$(dir2,2)
140   old_datad$=DATAD$:old_destd$=DESTD$
150   DATA_USE '':ERRno=-7
160   REPeat sloop
170     IF FTEST(dir1$)<0
180       dir1$=old_datad$&dir1$
190       IF FTEST(dir1$)<0:PRINT #0,dir1$;' ';:EXIT sloop
200     END IF
210     full_dir$=(dir1$&' ')(1 TO 5):orig_dir$=dir1$
220     IF FTEST(dir2$)<0
230       outer=FOP_NEW(dir2$):IF outer>0:CLOSE #outer
240       IF outer<0
250         dir2$=old_destd$&dir2$
260         IF old_destd$(LEN(old_destd$))<>'_':ERRno= -15:EXIT sloop
270         IF FOP_OVER(dir2$)<0:PRINT #0,dir2$;' ';:EXIT sloop
280       END IF
290     END IF
300     ERRno=0:EXIT sloop
310   END REPeat sloop
320   DATA_USE old_datad$
330   IF ERRno<0:REPORT ERRno:RETurn
340   IF dir2$(LEN(dir2$))<>'_':dir2$=dir2$&'_'
350   main_ch=-1:max_ch=0
360   read_directory dir1$
370   PRINT #0,'Backup complete'
380   FOR i=main_ch TO max_ch:CLOSE #i
```

(continues on next page)

(continued from previous page)

```
390 END DEFine
400 :
410 DEFine PROCedure read_directory(current_dir$)
420   LOCal in_ch,out_ch
430   in_ch=FOPEN('scr_'):IF main_ch=-1:main_ch=in_ch
440   out_ch=FOPEN(pipe_10000):DIR #out_ch,current_dir$
450   TCONNECT #out_ch TO #in_ch
460   CLOSE #out_ch
470   copy_file$ #in_ch,full_dir$,dir2$
480   IF in_ch>max_ch:max_ch=in_ch
490 END DEFine
500 :
510 DEFine PROCedure copy_file$(chan,in$,out$)
520   LOCal files_loop,junk$,outer,test1,test2
530   INPUT #chan,junk$,junk$
540   REPeat files_loop
550     IF EOF(#chan):EXIT files_loop
560     INPUT #chan,in_file$
570     out_file$=out$&in_file$
580     in_file$=in$&in_file$
590     IF LEN(in_file$)>3
600       IF in_file$(LEN(in_file$)-2 TO)=' ->'
610         read_directory in_file$(1 TO LEN(in_file$)-3)
620         NEXT files_loop
630       END IF
640     END IF
650     test1=FBKDT(\in_file$)
660     outer=FOPEN(out_file$)
670     IF outer>0
680       test2=FUPDT(#outer):CLOSE #outer
690     ELSE
700       test2=-7
710     END IF
720     IF test2<test1 OR test1=0
730       PRINT 'Backing-up'!in_file$!'=>'!out_file$
740       DELETE out_file$:COPY in_file$ TO out_file$
750       SET_FBKDT \in_file$,DATE
760     END IF
770   END REPeat files_loop
775   CLOSE#chan
780 END DEFine
```

### CROSS-REFERENCE

*FUPDT*, *FLEN*, *FTYP*, *FDAT*, *FXTRA*, *FILE_LEN*, *FILE_LEN*, *FILE_TYPE*, *FVERS* and *FNAME$* return other information about a file.

## 13.9 FDAT

| | |
|---|---|
| Syntax | FDAT [(#channel)] or<br>FDAT (\filename) (Toolkit II and THOR) |
| Location | Toolkit II, THOR XVI, BTool |

This function returns the value of four bytes (at offset 6 to 9) in a file header. This value represents the dataspace of executable files (file type 1). There is no convention for any other file types. The default data device and sub-directories are supported, the default channel is #3.

**CROSS-REFERENCE**

*FXTRA* returns the other four bytes of the type dependent information contained in the file header. *FILE_DAT* is very similar to *FDAT*. See also *FTYP*.

## 13.10 FDEC$

| | |
|---|---|
| Syntax | FDEC$ (value,length,ndp) |
| Location | Toolkit II, THOR XVI |

This function is similar to CDEC$ except for two major differences. FDEC$ does not assume that value is an integer, and therefore uses the whole of value, although if the given ndp (number of decimal places) is less than the number of decimal places in value, value will be rounded up or down accordingly.

FDEC$ does not insert commas in the characters to the left of the decimal point.

**Examples**

```
PRINT FDEC$(100.235,6,2)
```

will print '100.24'

```
PRINT FDEC$(100,6,2)
```

will print '100.00'

**CROSS-REFERENCE**

Please see *CDEC$*.

## 13.11 FETCH_BYTES

| | |
|---|---|
| Syntax | a$ = FETCH_BYTES(#channel, how_many) |
| Location | DJToolkit 1.16 |

This function returns the requested number of bytes from the given channel which must have been opened for INPUT or INPUT/OUTPUT. It will work on CON_ channels as well, but no cursor is shown and the characters typed in are not shown on the screen. If there is an ENTER character, or a CHR$(10), it will not signal the end of input. The function will not return until the appropriate number of bytes have been read.

WARNING - JM and AH ROMS will cause a 'Buffer overflow' error if more than 128 bytes are fetched, this is a fault with QDOS and not with DJToolkit. See the demos file, supplied with DJToolkit, for a workaround to this problem.

**EXAMPLE**

```
LineOfBytes$ = FETCH_BYTES(#4, 256)
```

## 13.12 FEXP$

| | |
|---|---|
| Syntax | FEXP$ (value,length,ndp) |
| Location | Toolkit II |

This function is different to CDEC$ in that it always prints the given value in exponential format. This means that there is always only one character to the left of the decimal point (plus any sign), and ndp (number of decimal places) states how many characters should be to the right of the decimal point.

FEXP$ does not assume that value is an integer and therefore also caters for floating point values. The length of the field must be at least ndp+7, otherwise an empty string is returned.

If necessary, values are rounded up or down to fit in the specified ndp number of decimal places.

**Examples**

```
PRINT FEXP$(-100.235,11,4)
```

will print -1.0023E+02

```
PRINT FEXP$$(100.235,11,4)
```

will print 1.0024E+02

**CROSS-REFERENCE**

*CDEC$*, *IDEC$*, *FDEC$* and *PRINT_USING* all provide means of formatting number output.

## 13.13 FET

| | |
|---|---|
| Syntax | FET(file *[,{filex \| #chx}]* [;cmd$]) |
| Location | SMSQ/E |

Executes and returns the job ID of the job filename in suspended state (by immediately setting the new job's priority to zero). This function is a functional version of *ET*, with the same set of arguments, that executes a job into suspended state for tracing with a monitor. In addition to what *ET* does, it returns the job ID of the new job that was started.

**Examples**

```
jId = FEX ("win1_XChange_xchange")
```

Will start Psion XChange in win1_xchange in suspended state and return the job ID of the new job.

**CROSS-REFERENCE**

See *ET*, *EXEC*, *JOBS*.

## 13.14 FEW

| | |
|---|---|
| Syntax | FEW(filen *[,{filex \| #chx}]* [;cmd$]) |
| Location | SMSQ/E |

Executes and waits for completion of the job filen, then returns the error code from that job. FEW is a function version of *EW* and shares its argument list.

**Example**

```
retcode = FEW ("win1_XChange_xchange")
```

Will start Psion XChange in win1_xchange, wait until that job has ended and will then return the error code of that job.

**CROSS-REFERENCE**

See *EW*, *EXEC_W*, *JOBS*, *QUIT*.

## 13.15 FEX

| | |
|---|---|
| Syntax | FEX(file *[,{filex \| #chx}]* [;cmd$]) |
| Location | SMSQ/E |

Executes and returns the job ID of the job filename. This function is a functional version of *EX*, with the same set of arguments, that executes a job. In addition to what *EX* does, it returns the job ID of the new job that was started.

**Example**

```
jId = FEX ("win1_XChange_xchange")
```

Will start Psion XChange in win1_xchange and return the job ID of the new job.

**NOTES** In some combinations of SMSQ/E and FileInfo2 versions, there might be a clash of extension names between FEX supplied as an SMSQ/E function (as described here) and a function with the same name (but very different purpose) supplied by FileInfo2. Later versions of FileInfo2 resolved this name clash by renaming the corresponding function to EXF.

**CROSS-REFERENCE**

See *EX*, *FET*, *FEW*, *EXEC*, *JOBS*.

## 13.16 FEX_M

| | |
|---|---|
| Syntax | FEX_M file *[,{file$^x$ \| #ch$^x$}]* [;cmd$] |
| Location | SMSQ/E |

Variant of the *FEX* function that executes the given file and returns the new job ID. Differently to *FEX*, which starts the new job as owned by the system, FEX_M starts the job as a job owned by its parent job. This means that the newly started job will be killed whenever its owner job is killed.

**Example**

```
the_job_id = FEX_M(win1_qmac)
```

Will execute Qmac as a job owned by the current S*BASIC interpreter. When the current interpreter ceases to exist, the new Qmac job will also be killed.

**NOTE**

An exhaustive explanation of the possible options can be found with the description of *EX*. FEX_M takes the exact same arguments.

**CROSS-REFERENCE**

See *EX*.

## 13.17 FF

| | |
|---|---|
| Syntax | FF |
| Location | Beuletools |

This function returns CHR$(12), which performs a form feed when sent to an EPSON compatible printer.

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*, *DBL*, *ENL*, *PRO*, *SI*, *NRM*, *UNL*, *ALT*, *ESC*, *LMAR*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 13.18 FGET%

| | |
|---|---|
| Syntax | FGET% [(#channel)] |
| Location | BTool |

This function reads two bytes from #channel (default #1) and makes an integer value from them, so these bytes should be in the internal format of an integer to make FGET% useful.

An integer is stored in two bytes as Integer = Byte1*256+ byte2

**CROSS-REFERENCE**

See *GET* and *MKI$*. *CVI%* converts a string containing the internal format of an integer to an integer number. See also *FPUT%*

## 13.19 FGET$

| | |
|---|---|
| Syntax | FGET$ [(#channel)] |
| Location | BTool |

This function reads a string in internal format from a specified channel (default #1) and returns the string.

A string is stored internally as a two byte integer (see FGET%) specifying the length of the string followed by the characters of the string itself.

**Example**

```
100 OPEN_NEW#3,ram1_test
110 PRINT#3,MKS$("Hello World.")
120 FPOS_A#3,0
130 PRINT FGET$(#3)
140 CLOSE#3
150 DELETE ram1_test
```

**CROSS-REFERENCE**

*GET*, *FGETB*, *FGET%*, *FGETL*, *MKS$*. *FPUT$* writes a string in internal format. *CVS$* converts a string into its internal format.

## 13.20 FGETB

| | |
|---|---|
| Syntax | FGETB [(#channel)] |
| Location | BTool |

This function reads a single byte (character) from a specified channel (default #1) and returns its numeric value.

**Example**

```
100 PRINT "Please press any key..."
110 CLEAR: c = FGETB
120 PRINT "You pressed '";CHR$(c);"', ";
130 PRINT "which is code"!c!"($";HEX$(c,8);")."
```

**CROSS-REFERENCE**

The Toolkit II equivalent is *BGET*. See also *FPUTB*!

## 13.21 FGETL

| | |
|---|---|
| Syntax | FGETL [(#channel)] |
| Location | BTool |

This function reads four bytes, being the internal representation of a longword, from a specified channel (default #1) and returns the longword's value.

**Example**

It is preferable to store a large integer in internal format because this is faster than text representation and needs less memory, even if the number could be stored in internal float format:

```
100 large_int = 1.19344E7
110 :
120 REMark save value
130 OPEN_NEW#3,ram1_test
140 PRINT#3,MKL$(large_int)
150 CLOSE#3: CLEAR
160 :
170 REMark read value
180 OPEN_IN#3,ram1_test
190 large_int = FGETL(#3)
200 CLOSE#3: PRINT large_int
```

**CROSS-REFERENCE**

*LGET*, *MKL$*. *FPUTL* allows you to write numbers in internal format to channels. *CVL* converts strings containing the internal format to long integers.

## 13.22 FGETF

| | |
|---|---|
| Syntax | FGETF [(#channel)] |
| Location | BTool |

The function FGETF gets six bytes from a channel (default #1) in the internal format of a floating point number.

**WARNING**

FGETF will hang SuperBASIC if the six bytes did not represent a valid floating point, so be careful.

**CROSS-REFERENCE**

*GET*, *MKF$*, *PEEK_F*, *FPUTF*. *CVF* converts a string containing the internal format into a floating point number.

## 13.23 FGETH$

| | |
|---|---|
| Syntax | FGETH$ [(#filechan)] |
| Location | BTool |

This function reads the file header from an open channel linked to a file (default #3).

Each file has a header of 64 bytes which contains technical information about the file. FGETH$ returns a string containing 64 characters, each of which represents one byte of the file header. The string contains the following information:

| Character | Meaning | Value in string | Equivalent Function |
|---|---|---|---|
| 1…4 | file length | CVL(h$(1 TO 4)) | FLEN |
| 5 | file access key | CODE(h$(5)) | None |
| 6 | file type | CODE(h$(6)) | FTYP |
| 7..14 | type dependent info (see below) | | FDAT,FXTRA |
| 15..16 | filename length | CVI%(h$(15 TO 16)) | LEN(FNAME$) |
| 17..52 | filename bytes | CVS$(h$(15 TO 52)) | FNAME$ |
| 53..56 | update time | CVL(h$(53 TO 56)) | FUPDT |
| 57..58 | version number | CVI%(h$(57 TO 58)) | FVERS |
| 59..60 | reserved | CVI%(h$(59 TO 60)) | None |
| 61..64 | backup date | CVL(h$(61 TO 64)) | FBKDT |

The type dependent information is different for each file type. For type 1 (executable files) bytes 7 to 10 hold the dataspace: CVL(h$(7 TO 10)). In early documentation, bytes 57 to 60 were reserved for a reference date which was never implemented. The last eight bytes (57 to 64) are actually not used on level-1 drivers, level-2 drivers use every byte. There is an *unofficial* standard for the file access key, which is generally used by Toolkits to store file attributes in the format:

| Bit No | Meaning |
|---|---|
| 7 | Set if the file is read-only. |
| 6 | Set if the file is hidden and will not appear on a directory of the disk. Neither can it be accessed. |
| 0 - 5 | are used to contain the User Number. Basically, this file will only be accessible by someone with the same user number (0-63).<br>Files with a user number of 0 will be visible and usable by any user.<br>Files with a user number of 63 are generally only available to a user in a special mode (normally this will require a password). |

You will need specialist toolkits such as Toolkit III and System, neither of which are compatible with SMS if the File Access Key is to have any effect.

**Examples**

Nearly every part of a file header (apart from the two unused bytes) can be read by special functions (see the list above), here are two functions to read the rest:

```
100 DEFine FuNction FACCKEY (chan)
110   LOCal h$
120   h$=FGETH$(#chan)
130   RETurn CODE(h$(5))
140 END DEFine FACCKEY
150 :
160 DEFine FuNction FSPEC% (chan)
170   LOCal h$
180   h$=FGETH$(#chan)
190   RETurn CVI%(h$(59 TO 60))
200 END DEFine FSPEC%
```

**CROSS-REFERENCE**

*FSETH$* is the counterpart of *FGETH$*. *HEADR* and *GetHEAD* read file headers to given memory positions, *FSETH$*, *HEADS* and *SetHEAD* set them. Functions like *FLEN*, *FTYP*, *FXTRA* etc. read the file header implicitly and return a certain piece of information from it. Use the *CVI%*, *CVL* and *CVS$* functions to convert the internal representations to actual values.

## 13.24 FILE_BACKUP

| | |
|---|---|
| Syntax | bk = FILE_BACKUP(#channel)<br>bk = FILE_BACKUP('filename') |
| Location | DJToolkit 1.16 |

This function reads the backup date from the file header and returns it into the variable bk. The parameter can either be a channel number for an open channel, or it can be the filename (in quotes) of a closed file. If the returned value is negative, it is a normal QDOS error code. If the value returned is positive, it can be converted to a string be calling DATE$(bk). In normal use, a files backup date is never set by QDOS, however, users who have WinBack or a similar backup utility program will see proper backup dates if the file has been backed up.

**EXAMPLE**

```
1000 bk = FILE_BACKUP('flp1_boot')
1010 IF bk <> 0 THEN
1020    PRINT "Flp1_boot was last backed up on " & DATE$(bk)
1030 ELSE
1040    PRINT "Flp1_boot doesn't appear to have been backed up yet."
1050 END IF
```

**CROSS-REFERENCE**

*FILE_DATASPACE*, *FILE_LENGTH*, *FILE_TYPE*, *FILE_UPDATE*.

## 13.25 FILE_DAT

| | |
|---|---|
| Syntax | FILE_DAT (filename) or<br>FILE_DAT (file$) |
| Location | TinyToolkit |

This is the same as FDAT except that default devices and sub- directories are not supported.

## 13.26 FILE_DATASPACE

| | |
|---|---|
| Syntax | ds = FILE_DATASPACE(#channel) or<br>ds = FILE_DATASPACE('filename') |
| Location | DJToolkit 1.16 |

This function returns the current dataspace requirements for the file opened as #channel or for the file which has the name given, in quotes, as filename. If the file is an EXEC'able file (See *FILE_TYPE*) then

the value returned will be the amount of dataspace that that program requires to run, if the file is not an EXEC'able file, the result is undefined, meaningless and probably zero. If the result is negative, there has been an error and the QDOS error code has been returned.

**EXAMPLE**

```
1000 ds = FILE_DATASPACE('flp1_WinBack_exe')
1010 IF ds <= 0 THEN
1020    PRINT "WinBack_exe doesn't appear to exist on flp1_, or is not␣
↪executable."
1030 ELSE
1040    PRINT "WinBack_exe's dataspace is set to " & ds & " bytes."
1050 END IF
```

**CROSS-REFERENCE**

*FILE_BACKUP*, *FILE_LENGTH*, *FILE_TYPE*, *FILE_UPDATE*.

## 13.27 FILE_LEN

| | |
|---|---|
| Syntax | FILE_LEN (filename) or<br>FILE_LEN (file$) |
| Location | TinyToolkit |

This function returns the length of a file in bytes. It does not support the default devices or sub-directories.

**Example**

A short program to show simple file statistics (without any support of wild cards):

```
100 dev$="FLP1_"
110 OPEN#3,PIPE_10000: OPEN#4,PIPE_200
120 TCONNECT #3 TO #4
130 DIR#3,dev$: INPUT#4,h$\h$
140 :
150 sum=0: count=0
160 REPeat add_lengths
170   IF NOT PEND(#4) THEN EXIT add_lengths
180   INPUT#4,file$
185   IF " ->" INSTR file$ THEN NEXT add_lengths
190   sum=sum+FILE_LEN(dev$ & file$)
200   count=count+1
210 END REPeat add_lengths
220 :
230 CLS
240 PRINT "There are"!count!"files in"!dev$;"."
250 PRINT "They are altogether"!sum!"bytes long,"
260 PRINT "the average length is"!INT(sum/count+.5)!"bytes."
```

TinyToolkit's TCONNECT or DIY Toolkit's QLINK is necessary

**NOTE**

It is not recommended to get a file list by writing a directory into a file or pipe. Problems arise with sub-directories on level-2 drivers: a sub-directory is marked with an appended " ->" in the directory list (WDIR, WSTAT, DIR), so opening a file such a "test ->" will fail. Refer to OPEN_DIR and FOP_DIR for a cleaner method.

**CROSS-REFERENCE**

*FLEN* has a more flexible syntax. *FILE_TYPE*, *FILE_DAT*, *FILE_POS*, *FNAME$*, *FPOS*, *FTYP*, *FUPDT* and *FXTRA* hold other information on a file.

## 13.28 FILE_LENGTH

| | |
|---|---|
| Syntax | fl = FILE_LENGTH(#channel)<br>fl = FILE_LENGTH('filename') |
| Location | DJToolkit 1.16 |

The file length is returned. The file may be open, in which case simply supply the channel number, or closed, supply the filename in quotes. If the returned value is negative, then it is a QDOS error code.

**EXAMPLE**

```
1000 fl = FILE_LENGTH('flp1_WinBack_exe')
1010 IF fl <= 0 THEN
1020    PRINT "Error checking FILE_LENGTH: " & fl
1030 ELSE
1040    PRINT "WinBack_exe's file size is " & fl & " bytes."
1050 END IF
```

**CROSS-REFERENCE**

*FILE_BACKUP*, *FILE_DATASPACE*, *FILE_TYPE*, *FILE_UPDATE*.

## 13.29 FILE_OPEN

| | |
|---|---|
| Syntax | FILE_OPEN ([#ch,] device [,{mode% \| ChID}]) |
| Location | BTool |

FILE_OPEN is a function which will open any device (default data directory supported for files) for all kinds of tasks. If a channel number #ch is not supplied, FILE_OPEN will choose the channel number on its own by searching for the next free channel number and returning it.

FILE_OPEN returns the channel number if it was not specified or otherwise zero. In case of failure it will return a (negative) error code. If error -4 ('out of range') is returned when a channel number has not been supplied, this indicates that the channel table of a compiled job is full.

The third parameter can be either the open mode or the channel ID of an un-named pipe.

The open mode (default 0) is:

- 0 (old exclusive) - open an existing file to read and write.
- 1 (old shared) - open an existing file to read only.

- 2 (new exclusive) - create a new file if it does not exist.
- 3 (new overwrite) - create a new file, whether or not it exists.
- 4 (dir open) - open a directory to read only.

If the third parameter is the channel ID of an open input pipe, then FILE_OPEN will create an output pipe linked to that channel.

**Example**

Count additional keywords:

```
100 ch1=FILE_OPEN(pipe_10000)
110 ch2=FILE_OPEN(pipe_,CHANID(#ch1))
120 EXTRAS#ch1
130 FOR count=1 TO 1E6
140   IF IO_PEND%(#ch2) THEN EXIT
150   INPUT#ch2,keyword$
160   AT 0,0: PRINT count
170 END FOR count
180 CLOSE#ch1,#ch2
```

**CROSS-REFERENCE**

*FILE_OPEN* combines *OPEN*, *OPEN_IN*, *OPEN_NEW*, *OPEN_OVER*, *OPEN_DIR*, *FOPEN*, *FOP_IN*, *FOP_OVER*, *FOP_NEW*, *FOP_DIR*, *TTEOPEN* and *TCONNECT*. See also *CHANID* and *ERNUM*.

## 13.30 FILE_POS

| | |
|---|---|
| Syntax | FILE_POS (#channel) |
| Location | TinyToolkit |

This performs the same function as *FPOS*, although with slightly less flexible parameters.

## 13.31 FILE_POSITION

| | |
|---|---|
| Syntax | where = FILE_POSITION(#channel) |
| Location | DJToolkit 1.16 |

This function will tell you exactly where you are in the file that has been opened, to a directory device, as #channel, if the result returned is negative it is a QDOS error code. If the file has just been opened, the result will be zero, if the file is at the very end, the result will be the same as calling FILE_LENGTH(#channel) - 1, files start at byte zero remember.

**EXAMPLE**

```
1500 DEFine FuNction OPEN_APPEND(f$)
1510   LOCal ch, fp
1515   :
```

(continues on next page)

(continued from previous page)

```
1520   REMark Open a file at the end, ready for additional
1530   REMark data to be appended.
1540   REMark Returns the channel number. (Or error)
1545   :
1550   ch = DJ_OPEN(f$)
1560   IF ch < 0 THEN
1570      PRINT "Error: " & ch & " Opening file: " & f$
1580      RETurn ch
1590   END IF
1595   :
1600   MOVE_POSITION #ch, 6e6
1610   fp = FILE_POSITION(#ch)
1620   IF fp < 0 THEN
1630      PRINT "Error: " & fp & " reading file position on: " & f$
1640      CLOSE #ch
1650      RETurn fp
1660   END IF
1665   :
1670   PRINT "File position set to EOF at: " & fp & " on file: " &f$
1680   RETurn ch
1690 END DEFine
```

**CROSS-REFERENCE**

*ABS_POSITION*, *MOVE_POSITION*.

## 13.32 FILE_PTRA

| | |
|---|---|
| Syntax | FILE_PTRA #channel, position |
| Location | TinyToolkit |

This command forces the file pointer to be set to the given position. Positions greater than the actual file length or smaller than zero will set the pointer to the end or start of the file respectively.

**CROSS-REFERENCE**

*FILE_PTRR*, *FILE_POS*, *FPOS*, *FLEN*, *FILE_LEN*, *GET*.

## 13.33 FILE_PTRR

| | |
|---|---|
| Syntax | FILE_PTRR #channel, bytes |
| Location | TinyToolkit |

This command moves the file pointer from its current position by the given number of bytes forward, negative numbers allow backward movement.

The file pointer cannot go beyond the limits of the file itself, so if you try to do so, the pointer will be set to the start or end of the file.

**Example**

A program to store several names and telephone numbers in a file and then to search for the given name and return the relevant telephone number:

```
100 DIM a$(3,30),number(3)
110 RESTORE
120 FOR i=1 TO 3: READ a$(i),number(i)
130 OPEN_NEW #3,flp2_phone_dbs
140 FOR stores=1 TO 3
150   PUT#3,a$(stores),number(stores)
160 END FOR stores
170 CLOSE#3
180 :
200 INPUT name$
210 OPEN_IN#3,flp2_phone_dbs
220 REPeat find_NAME
230   IF EOF(#3) THEN PRINT 'NAME not found...': STOP
240   GET#3,entry$
250   IF entry$==name$ THEN
260     GET#3,telno
270     EXIT find_NAME
280   END IF
290   FILE_PTRR#3,6: REMark skip next phone number
300 END REPeat find_NAME
310 CLOSE#3
320 PRINT entry$;'....';telno
330 :
350 DATA 'P.C. Green','999'
360 DATA 'CATFLAP inc.','7212.002121'
370 DATA 'Tim','98081'
```

Note that on Minerva, Integer Tokenisation will need to be disabled.

**CROSS-REFERENCE**

*FILE_PTRA*, *FILE_POS*, *FPOS*, *FLEN*, *FILE_LEN*, *GET*.

## 13.34 FILE_TYPE

| | |
|---|---|
| Syntax | ft = FILE_TYPE(#channel)<br>ft = FILE_TYPE('filename') |
| Location | DJToolkit 1.16 |

This function returns the files type byte. The various types currently known to me are :

- 0 = BASIC, CALL'able machine code, an extensions file or a DATA file.
- 1 = EXEC'able file.
- 2 = SROFF file used by linkers etc, a C68 Library file etc.
- 3 = THOR hard disc directory file. (I think!)

- 4 = A font file in The Painter
- 5 = A pattern file in The Painter
- 6 = A compressed MODE 4 screen in The Painter
- 11 = A compressed MODE 8 screen in The Painter
- 255 = Level 2 driver directory or sub-directory file, Miracle hard disc directory file.

There *may* be others.

**EXAMPLE**

```
1000 ft = FILE_TYPE('flp1_boot')
1010 IF ft <= 0 THEN
1020    PRINT "Error checking FILE_TYPE: " & ft
1030 ELSE
1040    PRINT "Flp1_boot's file type is " & ft & "."
1050 END IF
```

**CROSS-REFERENCE**

*FILE_BACKUP*, *FILE_DATASPACE*, *FILE_LENGTH*, *FILE_UPDATE*.

## 13.35 FILE_UPDATE

| | |
|---|---|
| Syntax | fu = FILE_UPDATE(#channel)<br>fu = FILE_UPDATE('filename') |
| Location | DJToolkit 1.16 |

This function returns the date that the appropriate file was last updated, either by printing to it, saving it or editing it using an editor etc. This date is set in all known QLs and emulators etc.

**EXAMPLE**

```
1000 fu = FILE_UPDATE('flp1_boot')
1010 IF fu <> 0 THEN
1020    PRINT "Flp1_boot was last written/saved/updated on " & DATE$(fu)
1030 ELSE
1040    PRINT "Cannot read lates UPDATE date from flp1_boot. Error: " & fu &
↪"."
1050 END IF
```

**CROSS-REFERENCE**

*FILE_DATASPACE*, *FILE_LENGTH*, *FILE_TYPE*, *FILE_TYPE*.

## 13.36 FILL

| | |
|---|---|
| Syntax | FILL [#channel,] boolean |
| Location | QL ROM |

This command switches Fill mode on and off. If the Fill mode is on (after FILL 1), all points in the given window channel (default #1) that have the same vertical co-ordinate are connected by a line in the current ink colour so that only non re-entrant figures can be filled correctly. This means that figures must only contain two points on each horizontal row of pixels. The fill mode is de-activated by FILL 0.

**Example 1**

```
FILL 1: POINT 20,20,40,20: FILL 0
```

draws a horizontal line from 20,20 to 40,20.

**Example 2**

```
100 DEFine PROCedure SQUARE (x,y,size,angle)
110   LOCal n: POINT x,y
120   TURNTO angle: PENDOWN: FILL 1
130   FOR n=1 TO 4: MOVE size: TURN 270
140   PENUP: FILL 0
150 END DEFine SQUARE
```

**NOTE 1**

FILL only affects those graphic commands which use relative co-ordinates, ie. which are influenced by SCALE. Commands which operate in absolute window or screen co-ordinates will not invoke filling.

**NOTE 2**

On non-Minerva ROMs, 1K of memory may be lost if you do not issue a FILL 0 before closing a window. This is however fixed by v1.38 (or later) of the Pointer Interface (although earlier versions will re-introduce it to Minerva!).

**NOTE 3**

When drawing several shapes, all of which are to be filled, ensure that you issue a FILL 0 between each shape, otherwise they will be joined together if any points appear on the same horizontal line!

**NOTE 4**

FILL works by setting aside a buffer of approximately 1K. Whenever a point is then plotted in the given window, FILL looks at the buffer to see if anything appears to the left of that point on the same horizontal line (in which case, it fills the line between the two points), otherwise, FILL will just note the co-ordinate of the point in its buffer.

FILL then checks if anything appears to the right of the given point, and if so, will fill the line between the two points. Again, the co-ordinate of the point will be stored if nothing appears to the right of it.

This should explain quite a few of FILL's quirks. Whenever a new FILL command is used on that window, the old buffer is lost, meaning that FILL will forget about any points previously plotted.

Unfortunately, the interaction of this buffer causes a lot of problems (and prevents re-entrant shapes), especially in view of the fact that only FILL or CLOSE will clear the buffer. The buffer is not cleared once a shape has been completely filled (eg. with CIRCLE), nor even when the screen is cleared with CLS. Try this for example:

```
100 INK 7:FILL 1
110 CIRCLE 50,50,20
130 CLS
```

(continues on next page)

(continued from previous page)

```
135 INK 2
140 CIRCLE 70,60,20
```

**NOTE 5**

If OVER -1 is switched on, the same line of an image may be FILLed twice causing that line to be left empty, unless you start drawing the image from either the top or the bottom. You may also encounter problems if you try to draw a line which has already been completed by FILL - for example try:

```
100 OVER -1: FILL 1
110 LINE 50,50 TO 60,60 TO 70,50 TO 50,50
```

The FILL command will complete the triangle as soon as the line between the points (60,60) and (70,50) has been drawn, therefore this should be re-written:

```
100 OVER -1:FILL 1
110 LINE 50,50 TO 60,60 TO 70,50
```

On Minerva v1.97 and SMSQ/E, matters are further complicated - the first example draws a complete triangle, whereas the second one doesn't!

**NOTE 6**

If OVER -1 is switched on, a shape which is drawn as FILLed will not be wiped out by re-drawing the same shape again, unless you do a FILL 1 before re-drawing the shape. For example, try this:

```
100 OVER -1:FILL 1:CIRCLE 50,50,20
110 PAUSE: CIRCLE 50,50,20
```

The answer is to insert a line:

```
105 FILL 1
```

**NOTE 7**

On Minvera pre v1.86 FILL 0 when fill was not actually switched on would stop SuperBASIC!!

**CROSS-REFERENCE**

The paint colour of *FILL* is specified by *INK*.

## 13.37 FILL$

| | |
|---|---|
| Syntax | FILL$ (short$,length) |
| Location | QL ROM |

This function will generate a string of the given length and return it. The new string will consist of a repeated series of short$ which may be one or two characters long. The length (as with any string) ranges from 0 to 32767.

**Examples**

```
FILL$("W-",7)
```

returns "W-W-W-W".

```
FILL$("+",10)
```

returns "++++++++++".

```
FILL$("Jo",0)
```

returns "" (the empty string).

```
FILL$("Test",6)
```

returns "TeTeTe".

**NOTE 1**

A bug in the THOR XVI (v6.40) meant that the return stack could be destroyed when appending the result to an even length string.

**NOTE 2**

A program will run more quickly (although it is more difficult to type in) if you declare the string explicitly rather than using FILL$.

**NOTE 3**

The maximum length of string that can be produced with FILL$ depends on the ROM version - see the Compatibility Appendix.

**CROSS-REFERENCE**

Refer to *DIM* about strings in general.

## 13.38 FILLMEM_B

| | |
|---|---|
| Syntax | FILLMEM_B start_address, how_many, value |
| Location | DJToolkit 1.16 |

Fill memory with a byte value. See *FILLMEM_L* below.

**CROSS-REFERENCE**

*FILLMEM_L*, *FILLMEM_W*.

## 13.39 FILLMEM_W

| | |
|---|---|
| Syntax | FILLMEM_W start_address, how_many, value |
| Location | DJToolkit 1.16 |

Fill memory with a 16 bit word value . See *FILLMEM_L* below.

**CROSS-REFERENCE**

*FILLMEM_L*, *FILLMEM_B*.

## 13.40 FILLMEM_L

| | |
|---|---|
| Syntax | FILLMEM_L start_address, how_many, value |
| Location | DJToolkit 1.16 |

Fill memory with a long (32 bit) value.

**EXAMPLE**

The screen memory is 32 kilobytes long. To fill it all black, try this:

```
1000 FILLMEM_B SCREEN_BASE(#0), 32 * 1024, 0
```

or this:

```
1010 FILLMEM_W SCREEN_BASE(#0), 16 * 1024, 0
```

or this:

```
1020 FILLMEM_L SCREEN_BASE(#0), 8 * 1024, 0
```

and the screen will change to all black. Note how the second parameter is halved each time? This is because there are half as many words as bytes and half as many longs as words.

The fastest is FILLMEM_L and the slowest is *FILLMEM_B*. When you use *FILLMEM_W* or FILLMEM_L you must make sure that the start_address is even or you will get a bad parameter error. *FILLMEM_B* does not care about its start_address being even or not.

*FILLMEM_B* truncates the value to the lowest 8 bits, *FILLMEM_W* to the lowest 16 bits and FILLMEM_L uses the lowest 32 bits of the value. Note that some values may be treated as negatives when *PEEK*'d back from memory. This is due to the QL treating words and long words as signed numbers.

**CROSS-REFERENCE**

*FILLMEM_B*, *FILLMEM_W*.

## 13.41 FIND

| | |
|---|---|
| Syntax | FIND (procfn$) |
| Location | BTool |

If procfn$ is the name of a machine code keyword (eg. "FILL$") then the function FIND returns the address where the definition is stored in memory.

If, however, procfn$ contains the name of a SuperBASIC PROCedure or FuNction then FIND will return the line number where the PROCedure or FuNction starts.

FIND returns 0 if the passed name is unknown.

**Example**

<ALT><r> requests a Procedure/Function name and calls Toolkit II's full screen editor accordingly:

```
ALTKEY "r","ED FIND('')"&CHR$(192)&CHR$(192)
```

**CROSS-REFERENCE**

*KEY_ADD*, *ELIS*, *NEW_NAME* Also see *FLIS*.

## 13.42 FLASH

| | |
|---|---|
| Syntax | FLASH [#ch,] switch |
| Location | QL ROM |

This command turns on or off flashing in the specified window channel (default #1). Switch can only have the values 0 (to enable flashing) and 1 (to turn flashing on).

This command will only have any effect in MODE 8.

If flashing is enabled, then any characters PRINTed to the given window afterwards will be shown to flash - it is first written out as normal, but then the parts of the character which would normally be shown in the current INK colour will alternate with the colour of the background.

The colour of the background can in fact be different for each row of pixels - this is calculated by the colour of the left-most pixel on each row for each character PRINTed.

**Example**

This short listing shows the effect of the FLASH command - note that the display is not actually changed back to its original form.

```
100 PAPER 2: INK 1
120 CSIZE 3,1: MODE 8: CLS
130 FOR i=0 TO 50: LINE 80+i,80 TO 15+i,10
140 INK 7: CURSOR 100,120
150 OVER 1: FLASH 1: PRINT 'This is flashing'
160 CSIZE 1,0: FLASH 0
```

**NOTE 1**

This command only affects characters PRINTed to the screen after the FLASH 1. There is no effect on graphics commands, or BLOCK or LINE.

**NOTE 2**

Spurious results may occur if you write over part of a flashing character (with OVER -1).

**NOTE 3**

This command does not work on the Amiga-QDOS Emulator or ST/QL Emulators.

**CROSS-REFERENCE**

Please also refer to *UNDER*, *OVER* and *PRINT*. *MODE* resets the *FLASH* mode to off.

## 13.43 FLEN

| | |
|---|---|
| Syntax | FLEN [(#channel)] or<br>FLEN (\file)(Toolkit II and THOR only) |
| Location | Toolkit II, THOR XVI, BTool |

This function returns the length of a file in bytes. If the second version is used, then Toolkit II's default data device and sub-directories will be supported, meaning that the command will consult the default data directory if necessary (see DATAD$).

If you use the first version however, you will first of all need to open a channel. If you do not supply a channel number, then the default used by the function is #3.

**NOTE 1**

The space on disks, cartridges, ramdisks and all other media where files can be stored is divided up into sectors, which are normally 512 bytes long. A file does not occupy the number of bytes returned by FLEN but a certain number of sectors for the contents of the file itself, a few bytes for the file header and the directory entries (sector map, etc). The total number of sectors which are occupied by the file data are:

```
sectors = 2 + CEIL(FLEN(\file)/512)
```

**NOTE 2**

If the second syntax does not work, update your Toolkit.

**CROSS-REFERENCE**

*FILE_LEN* has a slightly different syntax. *FILE_TYPE*, *FILE_DAT*, *FILE_POS*, *FNAME$*, *FPOS*, *FTYP*, *FUPDT* and *FXTRA* hold other information about a file. *HEADR* and *HEADS* allow you to directly access a file header.

## 13.44 FLIS

| | |
|---|---|
| Syntax | FLIS (procfn$) |
| Location | Tiny Toolkit |

If procfn$ is the name of a SuperBASIC PROCedure or FuNction then FLIS will return the line number where the PROCedure or FuNction is defined.

If however, it is a machine code keyword (eg. "FILL$") then the function FLIS will return 0.

If the name is not recognised the error 'Not Found' is reported.

**CROSS-REFERENCE**

*KEY_ADD*, *ELIS*, *NEW_NAME* Also see *FIND*.

## 13.45 FLP_DENSITY

| | |
|---|---|
| Syntax | FLP_DENSITY density (density = S, D, H or E)<br>FLP_DENSITY (SMSQ/E for QPC only) |
| Location | Gold Cards, SMS, SMSQ/E for QPC |

There are four types of floppy disk drives which can be connected to a QL with a Gold Card (or to other computers which are running a QL emulator). The command FLP_DENSITY sets the type for use with FORMAT:

| Sides | Density | Abbrev | Capacity | FLP_Density |
|---|---|---|---|---|
| Single | Double | SSDD | 360 Kb | S |
| Double | Double | DSDD | 720 Kb | D (Not QPC) |
| Double | High | DSHD | 1440 Kb | H |
| Double | Extra | DSED | 3240 Kb | E (Not QPC) |

Parameters other than the four letters S, D, H and E, (including several characters or several parameters) are not allowed.

**Examples**

```
FLP_DENSITY h
FLP_DENSITY 'D'
```

**NOTE 1**

Tests have shown that it is not always advisable to FORMAT a disk to a lower density than would otherwise be possible, for example a high density disk to double density. The result may be that the number of good sectors is less than could have been achieved by formatting a disk of the lower density.

During testing, an undamaged double density disk was formatted to 1440 sectors and a high density disk to 2880 sectors, but if the high density disk had been formatted to double density, eg. with:

```
FLP_DENSITY D : FORMAT flp1_
```

less than 1440 sectors might be good sectors.

You may also find that some disk drives which support the higher density will be unable to read these disks, since it will presume that they are FORMATted to their maximum density.

**NOTE 2**

Since FLP_DENSITY only has any affect during formatting, it should generally be avoided. This does not really matter because a disk is automatically formatted to the highest possible density and it would be a waste of money to use a HD disk as a DD disk.

**NOTE 3**

If a high or extra density disk is formatted on a system which does not support those capacities, it will be formatted to double density without any disadvantages. Such a disk does not cause problems when used with a Gold Card QL.

**NOTE 4**

A double density disk cannot be formatted to a higher density with HD drives - the Level-2 (or Level-3) device driver will automatically reduce a density which had been set at too high a figure by FLP_DENSITY, to the appropriate figure. An ED drive however can successfully format HD disks and even DD disks to high and extra density, but such disks may be unreliable, ie. data could be easily lost.

**NOTE 4**

High density is only supported on 3.5" disks, not 5.25" disks (widely used on MS/DOS systems). Extra density also only exists on 3.5" disks. QL DD and HD formatted disks have the same physical (but not software) format as MS/DOS and Atari TOS disks.

**NOTE 5**

High density and Extra density disks are much faster than double density disks, ED disks can even be as fast as slow hard disks.

**NOTE 6**

FLP_DENSITY overrides the in-built trial-and-error density detection which is slow for HD drives and even slower with ED drives. This can however cause problems when FORMATting DSDD disks - see FORMAT!

**NOTE 7**

On SQMS/E for QPC, the same code letters may be added (after a *) to the end of the medium name to force a particular density format. (For compatibility with older drivers, if the code letter is omitted after the *, single sided format is assumed).

- FORMAT 'FLP1_Disk23' Format at highest density or as specified by *FLP_DENSITY*.
- FORMAT 'FLP1_Disk24*' Format single sided
- FORMAT 'FLP1_Disk25*S' Format single sided
- FORMAT 'FLP1_Disk25*D' Format double sided, double density

Also, FLP_DENSITY on it's own resets automatic density selection.

**CROSS-REFERENCE**

The same effect as *FLP_DENSITY* can be achieved with a special *FORMAT* syntax. *FLP_TRACK* allows you to specify the number of tracks to be formatted onto a disk. *STAT* prints the name, good and free sectors of a medium. See also the The DMEDIUM_XXX functions, starting at *DMEDIUM_DENSITY*.

## 13.46 FLP_DRIVE

| | |
|---|---|
| Syntax | FLP_DRIVE drive%, drive$ |
| Location | SMSQ/E for QPC |

This changes the drive/image the floppy device is connected to.

**Example**

```
FLP_DRIVE 2,"C:\FLOPPY.IMG"
```

Now FLP2_ is assigned to the floppy image FLOPPY.IMG on the host computer's C:\ drive.

```
FLP_DRIVE 2,"B:\"
```

FLP2_ is assigned to the physical B:\ floppy drive of the host computer.

## 13.47 FLP_DRIVE$

| | |
|---|---|
| Syntax | drive$ = FLP_DRIVE$(drive%) |
| Location | SMSQ/E for QPC |

This reads back the current connection of the floppy device.

**Example**

```
PRINT FLP_DRIVE$(2)
```

will tell you the current setting for flp2_.

## 13.48 FLP_EXT

| | |
|---|---|
| Syntax | FLP_EXT |
| Location | Gold Cards |

If you use RES_128 or RES_SIZE to reset the computer to 128K memory any attempts to access the floppy disk drives can be haphazard, and can even crash the computer.

The command FLP_EXT resolves these problems and adds the following commands for use: RAM_USE, CACHE_ON, CACHE_OFF, SCR2DIS, SCR2EN, AUTO_TK2F1, AUTO_TK2F2, AUTO_DIS, FLP_JIGGLE, PAR_USE, FSERVE, NFS_USE, DEV_USE, DEV_LIST, DEV_USE$, DEV_NEXT, SDUMP, SDP_SET, SDP_KEY, SDP_DEV, PRT_USE, PRT_ABT, RES_128, RES_SIZE, PROT_DATE

**CROSS-REFERENCE**

See *RES_128* and *TK2_EXT*.

## 13.49 FLP_JIGGLE

| | |
|---|---|
| Syntax | FLP_JIGGLE [driveno,] flag |
| Location | Gold Cards |

There were originally various problems when using Mitsubishi ED disk drives with the Gold Card and so a fix was incorporated in both the Gold Card and Super Gold Card operating systems which forces the drive read/write head to make a number of rapid steps.

This can however cause problems with other ED disk drives (normally seen in the form of 'Not Found' or 'Bad or Changed Medium' errors.

It was therefore felt necessary to be able to enable or disable this feature at the users request (the default is to have the feature disabled).

To enable this feature set flag to 1, 0 will disable it.

If driveno is not specified, then the setting will be applied to all disk drives connected to the (Super) Gold Card and automatically stored so that it is available on power on.

If driveno is specified, then the setting will only apply to that specified disk drive and will be forgotten when the power is switched off.

**CROSS-REFERENCE**

See *FLP_STEP* and *FLP_START* which overcome various other problems with some disk drives.

## 13.50 FLP_SEC

| | |
|---|---|
| Syntax | FLP_SEC level |
| Location | Gold Cards, Trump Card, SMS, THORs, SMSQ/E for QPC |

The Gold Card, Trump Card and Thor range of computers, together with SMS provide a high standard of disk security, meaning that they are unlikely to fail to notice when a disk has been swapped over, and thereby try to write a file across two disks!

However, this level of security does affect the speed of disk access, as the system must check to see if the disk has been altered each time that it is written to.

The command FLP_SEC allows you to choose between three levels of security, the lowest of which (level 0) is still at least as secure as many other disk operating systems (such as MSDOS). The lower the level of security, the quicker disk access will be. The levels of security are as follows:-

**Security Level 0**
The disk system will only check to see if the disk has changed if a file is opened and the disk has stopped (ie. the disk light will have gone out) since the last time it was checked. The disk map is only updated when a file is closed (or flushed) and no other disk access has happened within half a second. Confusion can be expected on both read and write operations whenever a disk is changed whilst the disk light is still on or there are files open to the disk.

**Security Level 1**
The disk is checked each time that a file is opened, data is written to the disk, or the disk map is to be written; provided that the disk has stopped since the last time it was checked. The disk map is only updated when a file is closed (or flushed) and no other disk access has happened within half a second. The disk is not checked when anything is read from the disk, which can lead to confusion if a disk is changed whilst there are files still open.

**Security Level 2**
The disk is checked whenever a file is opened, data is written to or read from the disk, or the map is to be read or written to; provided that the disk has stopped since the last time that it was checked. The disk map and directory are updated and the slave buffers flushed every time that a file is closed (or flushed).

**SMS NOTE**

FLP_SEC has no effect - the security level is fixed at 2, the most secure.

**SMSQ/E for QPC NOTE**

FLP_SEC has no effect - the security level is fixed at 2, the most secure.

## 13.51 FLP_START

| | |
|---|---|
| Syntax | FLP_START time |
| Location | Gold Cards, Trump Card, THORs, ST/QL (level D.02+ drivers), SMS, SMSQ/E for QPC |

The disk system always tries to read data from a disk as soon as it can. However, when writing to a disk, it is necessary to ensure that the disk is running at full speed before any information is sent to it.

For relatively new drives, the default waiting time of 0.5 seconds should be enough to ensure that the disk is running at full speed.

The command FLP_START can be used for older disks to allow a longer run-up time. You will need to specify the time in 20ms units - some older drives may require a value of about 60.

**Example**

```
FLP_START 13
```

sets the start up time to 13 * 20ms (260ms) - this may suit the most recent 3.5" drives.

**NOTE**

FLP_START has no effect on either the QXL or QPC implementations of SMSQ and SMSQ/E.

**CROSS-REFERENCE**

You may also need to alter the stepping rate with *FLP_STEP*.

## 13.52 FLP_STEP

| | |
|---|---|
| Syntax | FLP_STEP [drive,] rate |
| Location | Disk Interfaces, Gold Cards, SMS, SMSQ/E for QPC |

The step rate enables the computer to known how quickly to step across tracks on the disk surface. Normally, this is automatically set to 3 milliseconds (ms) for 80 track disks and 6ms for 40 track disks, although if the system detects repeated errors on reading the disk, it will automatically slow the step rate.

Various old disk drives may require a slower stepping speed (you will generally know this from the noise the disk drive makes - it will make a repetitive knocking sound each time that the disk is accessed). You can do this by increasing the value specified by setting the rate using this command.

If drive is not specified, the new step rate is taken to apply to all disk drives connected to the system, otherwise, you can specify the number of the drive to which the new step rate is to apply.

**Examples**

```
FLP_STEP 12
```

Will produce quite a slow step rate for older drives.

```
FLP_STEP 2,12
```

Will produce a step rate of 12ms for the drive in FLP2_.

**NOTE 1**

The first, optional parameter may not be available on some interfaces.

**NOTE 2**

FLP_STEP has no effect on the QXL, QPC or Atari implementations of SMSQ and SMS.

**CROSS-REFERENCE**

*FLP_SEC* will alter the security setting for reading and writing to a disk. *FLP_START* may also be needed on older drives.

## 13.53 FLP_TRACK

| | |
|---|---|
| Syntax | FLP_TRACK tracks |
| Location | Gold Cards, Trump Card, THOR, ST/QL, SMS |

When a disk is formatted, the operating system will check to see if there are more than 55 tracks on the disk, and if so, will presume that it should be formatted to 80 tracks (otherwise it will presume the disk is to be formatted to 40 tracks).

The command FLP_TRACK allows you to override this setting, so that you can format a disk to, say, 75 tracks. FLP_TRACK 40 should be used as standard when a 40 track disk drive is attached to the system as this will prevent the system from trying to read track 55 (which does not exist!!), thus saving wear on the drive.

**Example**

```
FLP_TRACK 40
```

can be used on a standard DSDD 80 track disk to format it into a form readable on a 40 track drive.

**CROSS-REFERENCE**

*FLP_DENSITY* also affects how a disk is *FORMAT*ted.

## 13.54 FLP_USE

| | |
|---|---|
| Syntax | FLP_USE [device] |
| Location | Gold Cards, Trump Card, THORs, ST/QL, SMS, SMSQ/E for QPC |

Software which was written in the early days of the QL tended to assume that it would always be run from microdrive, and therefore included no facilities to alter the default devices used by the software.

You may even find some software was written on a non-standard disk system and assumed that disks would be accessed via FDK rather than the normal FLP.

The FLP_USE command allows you to use such software by making the FLP device emulate any other device. You merely need to supply a three letter parameter representing the name of the device which is to be emulated. Once you do this, the FLP device will no longer be recognised. If the device is not specified, then the system reverts to using FLP to access the disk drives.

**Example**

```
FLP_USE 'mdv'
```

will allow you to use software which would normally run from microdrive (unless it is copy protected!).

**CROSS-REFERENCE**

*RAM_USE*, *DEV_USE* and *WIN_USE* are very similar. *DMEDIUM_TYPE* can be used to find out the type of device which a name actually refers to. *DMEDIUM_NAME$* will return the default name of a device.

## 13.55 FLUSH

| | |
|---|---|
| Syntax | FLUSH [#ch] |
| Location | Toolkit II |

The command FLUSH forces all of the QL's temporary buffers attached to the specified channel (default #3) to be emptied into that channel. This will only work on channels attached to files, any other type of channel will return error -15 (bad parameter).

This command is necessary due to the use by QDOS of slave blocks whenever a file is opened. Data can be stored partly in the slave blocks to aid speed and when writing to a file, which will only be written to that file once the channel has been CLOSEd or the slave blocks have become full.

Because of this, there is always a danger that part of the data will be lost if there is a power failure or other accident. FLUSH helps you to avoid this.

**NOTE**

FLUSH will not work with Micro Peripherals disk drives. Nor can it be used to flush the Networks.

**CROSS-REFERENCE**

See *OPEN* and *CLOSE*.

## 13.56 FLUSH_CHANNEL

| | |
|---|---|
| Syntax | FLUSH_CHANNEL #channel |
| Location | DJToolkit 1.16 |

This procedure makes sure that all data written to the given channel number has been 'flushed' out to the appropriate device. This means that if a power cut occurs, then no data will be lost.

**EXAMPLE**

```
1000 DEFine PROCedure SaveSettings
1010   OPEN_OVER #3, "flp1_settings.cfg"
1020   FOR x = 1 to 100
1030     PRINT #3, Setting$(x), Value$(x)
1040   END FOR x
1050   FLUSH_CHANNEL #3
1060   CLOSE #3
1070 END DEFine
```

## 13.57 FMAKE_DIR

| | |
|---|---|
| Syntax | FMAKE_DIR (subdirectory) |
| Location | Level-2 Device Drivers |

This function will only work if Level-2 or Level-3 device drivers are available.

FMAKE_DIR is identical to MAKE_DIR except that it is a function and does not stop a program if an error occurs, instead it returns the code of the error concerned.

The following errors need some explanation:

- Error -9 (in use) : There is already a sub-directory with the same name;
- Error -8 (already exists) : File (not a sub-directory) exists already with that name;
- Error -15 (bad parameter) : Medium does not support sub-directories.

**NOTE 1**

If MAKE_DIR or FMAKE_DIR fail on a ramdisk, an old type ramdisk may have been loaded. There is no other way to activate the integral ramdisk other than by resetting the whole system.

**NOTE 2**

If error -15 occurs (ie. if you try to created a sub-directory on a medium where this is not possible), MAKE_DIR and FMAKE_DIR will leave an empty file with the name of the desired sub-directory on the medium. Remember to remove this.

**CROSS-REFERENCE**

See *MAKE_DIR*.

## 13.58 FNAME$

| | |
|---|---|
| Syntax \| FNAME$ [(#channel)] or<br>FNAME$ (\file)(Toolkit II only) | |
| Location | Toolkit II, BTool |

This function returns the filename of a file attached to the specified channel (default #3), including the sub-directory prefix but without the pure device name (eg. RAM1_).

The second syntax enables you to find out the full filename of the specified file.

It is hard to understand why one should need to find out about the name of an opened file - the second syntax is even more absurd.

One possible usage is to convert a Toolkit II filename, which need not include the current sub-directory, to a full file name. However, the functions DATAD$, PROGD$ together with some string operations are much faster and more elegant because they skip the need to access the file header.

**CROSS-REFERENCE**

*FLEN*, *FTYP*, *FDAT*, *FXTRA*, *FUPDT*, *FILE_LEN* and *FILE_TYPE* return other information about a file.

## 13.59 FOPEN

| | |
|---|---|
| Syntax | FOPEN (#ch, name) or<br>FOPEN (name) |
| Location | Toolkit II, THOR XVI |

This function is designed to allow you to access files safely without causing errors which force a program to stop.

If the first variant of FOPEN is used, this is actually very similar to the command OPEN in operation, except that if for some reason opening the specified channel (#ch) with the specified name would cause an error, FOPEN returns the relevant error code. If the specified channel is successfully opened, then FOPEN returns 0.

By contrast, if the second variant of the command is used, where no specific channel number is used, if successful, FOPEN will return a positive number representing the number of the next available channel (ie. the number of the lowest entry in the channel table which is empty).

If a negative number is returned, this is the appropriate error number, allowing the programmer to take any necessary action (such as requesting the user to input a new file name).

**Examples**

```
ERRno = FOPEN(#3,scr_448x200a32x16)
Chan = FOPEN('flp1_Input_dat'): IF Chan>0 THEN INPUT #Chan,x
```

**NOTE 1**

All versions of this command (other than v2.28 of Toolkit II or later) can be confused by filenames which exceed 36 characters, in which case FOPEN will return 0. On later versions, FOPEN supports 41 character filenames (including any default directory).

**NOTE 2**

Although FOPEN opens a file for both reading and writing, it will only return an error if the file does not already exist or is in use. It does not check whether the file is read only. Use FOP_NEW or DMEDIUM_RDONLY for this. If you do not check whether the file is read only, an error will only be reported if you try to write to the file!!

**CROSS-REFERENCE**

*ERNUM* contains details of the various error messages. *WHEN ERRor* allows you to error trap a complete program. Also see *FOP_DIR*, *FOP_IN*, *FOP_OVER* and *FOP_NEW*. Also see *OPEN*. *FTEST* allows you to test the status of a file without (explicitly) opening a channel.

## 13.60 FOP_DIR

| | |
|---|---|
| Syntax | FOP_DIR (#ch, name) or<br>FOP_DIR (name) |
| Location | Toolkit II, THOR XVI |

The function FOP_DIR is a complementary function to OPEN_DIR in much the same way as FOPEN is to OPEN. This function returns the same values and suffers from the same problem as FOPEN.

**CROSS-REFERENCE**

See *FOPEN*, *TTEOPEN* and *OPEN_DIR*.

## 13.61 FOP_IN

| | |
|---|---|
| Syntax | FOP_IN (#ch, name) or<br>FOP_IN (name) |
| Location | Toolkit II, THOR XVI |

The function FOP_IN falls into the same series of functions as FOPEN, FOP_DIR, FOP_NEW and FOP_OVER. This function is a complementary function to OPEN_IN in much the same way as FOPEN is to OPEN. This function returns the same values and suffers from the same problem as FOPEN.

**CROSS-REFERENCE**

See *FOPEN* and *OPEN_IN*.

## 13.62 FOP_NEW

| | |
|---|---|
| Syntax | FOP_NEW (#ch, name) or<br>FOP_NEW (name) |
| Location | Toolkit II, THOR XVI |

This function, together with its companions FOPEN, FOP_IN, FOP_DIR and FOP_OVER, is designed to allow you to access files safely without causing errors which force a program to stop. This function is the complement to OPEN_NEW and returns the same values and suffers from the same problem as FOPEN. If the specified file already exists, you are asked whether you want to over-write the existing file. An error (-8) is returned if you press N, and error (-20) is returned if the disk is read only.

**CROSS-REFERENCE**

See *FOPEN* and *OPEN_NEW*.

## 13.63 FOP_OVER

| | |
|---|---|
| Syntax | FOP_OVER (#ch, name) or<br>FOP_OVER (name) |
| Location | Toolkit II, THOR XVI |

This function is the complement to OPEN_OVER and suffers from the same problem as FOPEN. It also returns the same values as FOP_NEW, except that it will implicitly over-write an existing file with the same name.

**CROSS-REFERENCE**

See *FOPEN* and *OPEN_OVER*.

## 13.64 FOR

| | |
|---|---|
| Syntax | FOR var = range *[,range$^{i}$]* |
| Location | QL ROM |

The SuperBASIC version of the classic FOR loop is *extremely* flexible.

The syntax of this SuperBASIC structure can take two forms:

FOR var=range *[,range$^{i}$]* :statement *[:statement]*

or :

FOR var=range *[,range$^{i}$]* *[statements]* [EXIT var] [NEXT var] END FOR var

Where range can be one of the following:

start_value TO end_value [STEP step]

or, simply just:

value

The first of these variants is known as an in-line FOR loop. Provided that there is at least one statement following FOR, this line will be repeated until the end value is reached (see below). There is no need for a related END FOR statement and therefore the shortest in-line FOR loop possible is:

```
FOR x=1 to 100: NEXT x
```

If an in-line loop is terminated prematurely, for example with EXIT, control will be passed to the statement following the corresponding END FOR statement (if one exists), or the next program line. This allows the following:

```
FOR x=1 TO 100: IF INKEY$=' ': EXIT x: END FOR x: PRINT x
```

The basic function of FOR is to count a floating point variable from a given start value to an end value by adding step to var during each pass of the loop (step may be positive or negative depending on the start and end values). If no step is specified, STEP 1 will be assumed.

However, if step is negative when the end value is greater than the start value (or vice versa), then the loop will immediately exit, and nothing contained in the loop will be processed.

A similar effect can be achieved by using a REPeat structure:

```
var=start_value
REPeat loop
  ...
  IF var >= end_value THEN
    EXIT loop
  ELSE var = var + step
END REPeat loop
```

The similarity between these two SuperBASIC loop types can be extended to the use of EXIT and NEXT statements which can be used identically in both structures.

EXIT terminates the loop, and the next statement which will be processed is the first statement after the corresponding END FOR. NEXT forces the program to make the next pass of the loop.

**PROGRAMMING NOTES**

1. When NEXT is used within a FOR..END FOR structure, if var is already at the end_value, the NEXT statement will have no effect:

```
100 FOR x=1 TO 9
110 PRINT x;" ";
120 IF x MOD 2 THEN NEXT x
130 PRINT x^2
140 END FOR x
```

Output:

```
1 2 4
3 4 16
5 6 36
7 8 64
9 81
```

To prevent the odd result when x=9, line 120 would need to be altered to read:

```
120 IF x MOD 2 THEN NEXT x: EXIT x
```

2. Except on a Minerva ROM or under SMS, the loop variable is set to 0 before the FOR is executed, therefore the following program prints the square roots of the numbers 0 to 9:

```
100 x=3
110 FOR x=x TO 9
120 PRINT x;' ';
130 IF NOT RND(10) THEN EXIT x
140 PRINT SQRT(x)
150 END FOR x
```

On Minerva ROMs and under SMS, this would print out all of the square roots of the numbers 3 to 9 (as expected).

3. A NEXT statement directly after the FOR statement could be used to omit some values of the loop variable:

```
100 FOR x=1 TO 9
110 IF x MOD 2 THEN NEXT x: EXIT x
120 PRINT x; TO 4; x^2
130 END FOR x
```

However, in some cases, it may be easier and shorter to write:

```
100 FOR x=2,4,6,8
110 PRINT x; TO 4; x^2
120 END FOR x
```

4. Single values and intervals can be freely mixed after the equals sign. The following examples are all valid expressions:

```
FOR x=2,4 TO 10 STEP 2,4.5,7 TO -4 STEP -.2
FOR x=1
```

5. To shorten program lines even further, the FOR loop can be used in a single line and the END FOR omitted (this is called an in-line FOR loop):

```
FOR x=2,4,6,8: PRINT x; TO 4; x^2
```

**Example 1**

A short routine to count the lines of a text file (using the oddities of the NEXT command):

```
100 OPEN#3,file
110 FOR lines=0 TO 10000
120   IF EOF(#3) THEN PRINT lines: EXIT lines
130   INPUT #3,line$: NEXT lines
140   PRINT 'OOPS - program is longer than 10000 lines!!'
150 END FOR lines
160 CLOSE#3
```

**Example 2**

The next example is a routine to nest a variable number (loops) of times which go from Value_min to Value_max at Value_step:

```
100 FOR loop=1 TO loops:Value(loop)=Value_min(loop)
110 REPeat Nesting
120 <instructions using Value(1...s) go here>
130   FOR loop=1 TO loops
140     IF Value(loop)=Value_max(loop) THEN
150       IF loop=loops THEN EXIT Nesting
160       Value(loop)=Value_min(loop)
170       NEXT loop
180     ELSE
190       Value(loop)=Value(loop)+Value_step(loop)
200       EXIT loop
210     END IF
220   END FOR loop
230 END REPeat Nesting
```

**NOTE 1**

If you use multiple in-line FOR loops in the same program line, only the inner loop will be executed. For example:

```
FOR i=1 TO 3: FOR j=1 TO 10: PRINT i*j: END FOR j
```

Output:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

This will actually work correctly under SMS.

You can get it to work on a Minerva ROM and under SMS (but not others) if the line is amended to read:

```
FOR i=1 TO 3: FOR j=1 TO 10: PRINT i*j: END FOR j: END FOR i
```

In fact, SMS will even allow the line to work if it simply reads:

```
FOR i=1 TO 3: FOR j=1 TO 10: PRINT i*j
```

**NOTE 2**

Unless you have SMS or a Minerva ROM, do not use GO SUB together with an in-line FOR loop, because this will act as an END FOR command and will not call the desired routine:

```
100 FOR i=1 TO 10: PRINT 'Junk - test';: GO SUB 200: PRINT i
110 STOP
200 PRINT ' Number ';
210 RETurn
```

**NOTE 3**

On JS (except ST/QL) and MGx ROMs, you cannot use the first of several PROCedure/FuNction parameters as the loop identifier:

```
100 TEST 5,10
110 FOR j=1 TO 10: PRINT 'OOPS...'
120 FOR k=1 TO 10: PRINT 'OKAY...'
125 :
130 DEFine PROCedure TEST(j,k)
140   AT j,k:PRINT 'Errors .....'
150 END DEFine
```

**NOTE 4**

No error will be reported and all should work okay if NEXT is used instead of END FOR (unless you try to use EXIT which would try to jump to the statement after the non-existent corresponding END FOR, and may reach the end of the program without finding the END FOR, therefore stopping without reporting any error), but you will have seen that NEXT is intended for another purpose.

Apart from programming elegance, compilers may not be able to understand your meaning (they assume that you have forgotten the END FOR) and may abort compilation or report a warning.

**NOTE 5**

Counting downwards without a negative step has no effect at all For example:

```
FOR loop=0 TO -3
```

Omitting the STEP parameter is the same as STEP 1.

**MINERVA NOTES**

On a Minerva machine, a FOR loop can use either a single character string variable or an integer variable:

```
FOR A$='A' TO 'Z' STEP CHR$(2):PRINT A$;' ';
```

This prints out:

```
A C E G I K M O Q S U W Y
```

```
FOR loop%=1 TO 255: ...: END FOR loop%
```

This is a little quicker than:

```
FOR loop=1 to 255: ...: END FOR loop
```

These examples will not work on other ROMs, unless you have SMS, even if they will let you type them in!

**SMS NOTES**

Like Minerva, SMS will allow you to use integer variables in FOR loops (but not string variables). As from v2.57, the range is checked to ensure that it is within the valid word integer range (-327678..32767) when the FOR loop is started, otherwise it returns 'Error in Expression'.

If you try to use a string loop variable, the error 'unacceptable loop variable' is reported. EXIT, NEXT and END FOR do not need to contain the loop identifier, SMS will presume that when used in a program, they refer to the loop currently being executed.

**CROSS-REFERENCE**

*REPeat* ... *END REPeat* is the other loop type. See also *END FOR*.

## 13.65 FORCE_TYPE

| | |
|---|---|
| Syntax | FORCE_TYPE string$ |
| Location | TinyToolkit |

This command forces the given string to be typed into the current keyboard queue, just as if you had typed it from the keyboard. There is not much use for this command in connection with applications because key macros such as ALTKEY are much easier to use. But, FORCE_TYPE can be used to perform an action without anyone actually needing to press a key.

**Example**

Your telephone rings and you talk half an hour with a friend. Meanwhile your computer crashes - God only knows why - and the BASIC program you were writing has now disappeared along with everything else.

You could decrease this danger by writing and compiling such a program:

```
100 last_stroke = DATE
110 REPeat Sleeping
120   IF KEYROW(1) THEN last_stroke = DATE
130   IF DATE-last_stroke > 300 THEN
140     FORCE_TYPE "SAVE_O FLP1_Backup_bas" & CHR$(10)
150     REPeat Wait: IF KEYROW(1) THEN EXIT Wait
160     last_stroke=DATE
170   END IF
180 END REPeat Sleeping
```

This example should obviously be adapted to your specific needs, applications and tools.

**NOTE 1**

Every console channel (ie. con_ windows) has a keyboard queue - the channel accessed by FORCE_TYPE must first be activated by a dummy INKEY$, PEND etc. to that channel.

**NOTE 2**

In earlier versions of Tinytoolkit (pre v1.10), this command was called TYPE_IN, which could cause problems with Turbo compiled programs.

**CROSS-REFERENCE**

*STAMP* does exactly the same as *FORCE_TYPE*.

## 13.66 FORMAT

| | |
|---|---|
| Syntax | FORMAT [#channel,] medium |
| Location | QL ROM |

Each medium where data can be stored as files (disks, ramdisks, microdrives or hard disks) has to be given a structure which is recognisable by QDOS. This is done by FORMATting it. Each medium can also be given a name of up to ten characters long. The command FORMAT clears a medium from scratch so that any data stored there is definitively lost. Be careful!

The following standard devices can be formatted:

- MDV1_ .. MDV8_ - microdrive cartridges
- FLP1_ .. FLP8_ - floppy disks
- RAM1_ .. RAM8_ - ramdisks
- WIN1_ .. WIN8_ - hard disks

Depending on the type of medium, several additions to the pure medium name are possible:

**MDV** Up to ten characters can be added, these will form the name of the cartridge, eg:

```
FORMAT mdv2_SuperBASIC
```

**FLP** As with microdrive cartridges, a medium name can be added. If the eleventh character of the name is an asterisk (*), the disk will be formatted single sided, ie. just the first side is used. In order to use the single sided only option, is it necessary to put the whole parameter in quotes, eg:

```
FORMAT "flp1_TEST *"
```

This is not applicable to HD and ED disks: their density will also be affected, making them single sided double density (SSDD). If a single sided disk can still be bought today, it will actually be a double sided disk of low quality.

With Super Gold Card, Gold Card and SMS, an appended asterisk plus a letter which indicates the density will format the disk accordingly: S, D, H and E are allowed, eg:

```
FORMAT "flp1_TEST*h"
```

See *FLP_DENSITY*.

**RAM** This depends very much on the ramdisk drivers:

With standard static ramdisks, which are built into most disk interfaces and available as public domain, you need to specify how many sectors are to be allocated to the ramdisk by adding the number of sectors to the device name, eg:

```
FORMAT ram1_200
```

formats ram2_ to 200 sectors (100K).

These static ramdisks must be FORMATted before use.

On the other hand, the Qjump ramprt ramdisk (provided with Qpac 2 and various expansion boards, including Trump Card, Gold Card and Super Gold Card) is dynamic - it adapts its size automatically to the size of the files being stored on it - there is no need to FORMAT the ramdisk prior to use. This can however also be used as a static ramdisk.

Trump Card, Super Gold Card and Gold Card ROMs also contain a special variant of a ramdisk which allows you to produce an image of a microdrive cartridge on a ramdisk, for example by using:

```
FORMAT ram4_mdv2.
```

Faulty files are marked with an asterisk added to the end of their filenames. Although this may allow you to 'rescue' a corrupt microdrive cartridge those files marked with an asterisk are faulty and therefore unreliable.

The name of a ramdisk is always the name of the medium without an underscore, eg. RAM1 for RAM1_; this is the same on dynamic ramdisks.

**WIN** A medium name can normally be stated, as with a microdrive cartridge. Please check the documentation of the hard disk drivers, they differ very much! For example, the firmware on the Falkenberg interface disables FORMAT for hard disks until certain settings have been specified with another command.

On the THOR, an asterisk needs to be included, eg:

```
FORMAT 'win1_*HARDDISK'
```

See below.

SMS for ATARI computers and QXL / QPC, expects you to have already partitioned the hard disk using the computer's native commands. On ATARIs, under SMSQ/E you then need to identify the drive and partition using WIN_DRIVE. After that, you can use the normal QL FORMAT command on all these systems, however, SMSQ/E has adopted a level of protection which insists that you must use the WIN_FORMAT command before FORMAT and the FORMAT command itself will display two characters on screen and ask you to type them in.

You should then use WIN_FORMAT to protect the partition again.

The standard drivers for the ST/QL Emulators adopt a form of protection in that you will need to type in the two characters shown on screen as with SMSQ/E.

You can also only FORMAT a hard disk from SuperBasic Job 0 and then only when Channel #0 is OPEN.

If the hard disk has already been partitioned by the Atari ST (the first partition will normally be marked GEM or BGM), then you will be asked to enter the number of the first partition to be used by QDOS and the number of subsequent partitions ot be used for this disk.

Under SMSQ/E on the QXL or QPC, this same two- level protection is adopted. However, instead of passing the medium name of the hard-disk, you have to pass the size of the QL hard disk to be created in megabytes, for example:

```
FORMAT WIN1_20
```

This will create a 20 Megabyte hard disk on PC drive C:

On early versions, the maximum size that could be created was 23 Megabytes and only one drive could be created. Later versions allow you to create WIN1 to WIN8 (all on drive C:).

After formatting, FORMAT will either report that the process has failed (error -14), because there was no cartridge/disk in the drive or if the medium was faulty.

The command will also fail if the given device was write-protected.

If everything was okay, a small message is printed to the specified channel (default #1) indicating how many sectors could be achieved and how many were good. If the two numbers differ, QDOS will have marked some sectors as bad and will ignore them. However, experience shows that if the difference between the two numbers is great, it can be very dangerous to store important data on those disks/cartridges.

It is recommended that new microdrive cartridges should be formatted 10 times before use (you should expect to get about 220 available sectors). It may also be useful to try formatting the cartridge in the other microdrive.

**Examples**

```
FORMAT mdv2_Startup
```

formats cartridge in microdrive 2

```
FORMAT "mdv2_Startup"
```

as above.

```
FORMAT flp1_backup
```

formats disk in disk drive 1

```
FORMAT "flp1_backup *"
```

as above but single sided

```
FORMAT "flp1_backup*d"
```

double sided, double density

```
FORMAT "flp2_backup*h"
```

double sided, high density

```
FORMAT ram1_100
```

format ramdisk 1 to 50K

```
FORMAT ram1_
```

remove ramdisk 1

```
FORMAT ram1_mdv1
```

format ram1_ to 255 sectors and copy cartridge in microdrive 1

**Notes on the different media:**

The traditional microdrive is relatively slow and unreliable, and cartridges need to be formatted several times to give good results (usually around 210-220 sectors) - pushing them firmly into the microdrive slot while they are being formatted is said to be more efficient.

However, as new cartridges are becoming more rare and expensive today, the next best and very highly recommended upgrade are disk drives.

It is also becoming less and less common to find users who can read information stored on microdrive, SMS and emulators for example, do not support microdrives.

**3.5" double density disks (720K)**

These are pretty cheap and you can get them everywhere (although the quality does vary); they have become a standard on the QL, although it is becoming ever more difficult to find replacement disk drives. FORMAT should report 1440 sectors.

**3.5" high density disks (1.4Mb)**

These are also fairly cheap and you can get them everywhere (although the quality does vary). These have become the new standard disks used by IBM compatible computers and therefore the disk drives are easy to obtain. FORMAT should report 2880 sectors.

**3.5" extra density disks (3.2Mb)**

These are fairly expensive and difficult to obtain as they were never really accepted in the IBM PC world, although for a time, they looked like becoming a new standard for the QL, being very quick and storing a lot of information. FORMAT should report either 1600 or 6400 sectors (see note 8).

**5.25" disks (720K)**

These are also widely spread in the QL scene, especially in the USA, although they are now becoming less and less common. With the introduction of the Super Gold Card and Gold Card by Miracle Systems Ltd, high density (HD, 1440K) and even extra density (ED, 3200K) drives have become available to QDOS for the first time. These formats are several times faster and even more reliable, not to mention the increased space for programs and data.

**Hard disks**

These are becoming more and more common, with them being readily available to people using Emulators on other computers, and also now the release of relatively cheap interfaces and disk drives for the QL and AURORA.

**Ramdisks**

These are not specific to any hardware configuration because they only exist in RAM and any stored data is lost if the machine is reset or turned off. On the other hand, ramdisks are extremely fast.

**NOTE 1**

Unless you have a Minerva ROM (see below), do not try to FORMAT a microdrive whilst any microdrive is still running, since this will report an 'in use' error.

```
PEEK(SYS_VARS+HEX('EE'))
```

will be zero if no microdrives are running.

**NOTE 2**

On the THOR XVI (v6.37 and earlier), there existed a bug when accessing anything greater than win2_.

**NOTE 3**

If there is no disk in a drive, FORMAT may also fail with a read only error (-20) instead of reporting not found (-7).

**NOTE 4**

You cannot use FORMAT n1_flp1_ (for example) to FORMAT a medium over the network.

**NOTE 5**

The ST/QL drivers cannot FORMAT the fifth and subsequent partitions on the hard disk unless the extended partition table is in the form used by SUPRA, ICD and similar drives.

**NOTE 6**

Minerva (pre v1.98) had some bugs in the code for FORMATting microdrives.

**NOTE 7**

FORMAT expects the specified channel (or #1) to be OPEN, otherwise an error will be reported.

**NOTE 8**

FORMAT cannot report a number of sectors in excess of 32768 and so may return wrong values on large capacity drives. SMS correctly reports the number of sectors obtained, although on an ED disk, FORMAT will report 1600 Sectors (DIR will show the figure of 6400 sectors instead!). This is because on an ED disk, sectors are 2048 bytes long instead of the usual 512 bytes expected by the QL device drivers (which have to be fooled to see each sector as 4x512 byte sectors).

**MINERVA NOTE**

On Minerva v1.78 (and later), a check is carried out before performing FORMAT to see if there are any files open on the desired medium. This stops Digital Precision's Conqueror and Solution from working correctly. To switch it off, use:

```
POKE !124 !49, PEEK (!124 !49) || 128
```

**SMS NOTES**

As with Minerva, you cannot FORMAT a medium if there are any files open on that medium ('Is In Use' error is reported). If there is a problem during the FORMAT process, SMS will emit a series of BEEPs. However, be warned that an error message is not always displayed and the FORMAT may appear to have completed correctly!! SMS does not allow you to access the QL's microdrives, nor can it solve the problem on the QXL below. SMS can corrupt floppy disks (so they have to be thrown away) if you try to FORMAT them to the wrong density.

Some users have reported problems in using SMS to FORMAT Double Density disks in ED disk drives linked to a Super Gold Card. This appears to afflict versions of SMS after v2.85 and all makes of ED drives. The problem only occurs if you specify the density with FORMAT 'flp1_NAME*D' or

FLP_DENSITY 'D'. In these cases, a noise is emitted during FORMAT to indicate that it has failed, but SMS still reports 1440/1440 sectors, even though subsequent attempts to access the disk report 'Not Found'. The answer is to not use FLP_DENSITY in this instance.

**QXL NOTES**

You cannot reliably FORMAT floppy disks from scratch on most PCs using this emulator. FORMAT merely re-formats an already formatted disk. Prior to v2.67 of SMS there existed several further problems with FORMAT on QXL.

**THOR XVI NOTES**

The THOR XVI, v6.37 (and later) allows a variant of the medium name to deal with the THOR's hard disk:

```
FORMAT "win1_options*name"
```

The available options which can be specified are:

- /C : Certify drive before formatting - this reconstructs the THOR's defect list, describing the bad sectors and tracks;
- /Q : Quick reformat - merely sets up new directory map;
- /F : Fast reformat - does not verify the disk;
- /Gn : Set group or cluster size in blocks. Default = /G16;
- /Dn : Set directory size in number of groups or clusters. Default = /D2.

**Examples**

```
FORMAT 'win1_/Q*Main'
FORMAT 'win1_/G16/D2*THORDisk'
```

**WARNING**

Prior to v2.71 of SMS FORMAT flp3_1 on the QXL could in fact FORMAT WIN1_.

**CROSS-REFERENCE**

Before formatting, the number of tracks on a disk can be specified with *FLP_TRACK*. HD and ED disks can be formatted to different densities if *FLP_DENSITY* was used to override automatic detection of the density. See *WIN_FORMAT* for hard disk protection. The DMEDIUM_XXX functions, starting at *DMEDIUM_DENSITY* return various details about how a medium has been formatted.

## 13.67 FPOS

| | |
|---|---|
| Syntax | FPOS [(#channel)] |
| Location | Toolkit II, THOR XVI, BTool |

This function returns the current position of the file pointer.

The relevant file must already be open as #channel, default channel is #3. A value of zero means that the file pointer is at the very beginning of a file, whereas a position equivalent to the file length means that it points to the very end. The file pointer is a means by which the QL can keep track of exactly whereabouts in a file it should take the next input from, or write to.

**CROSS-REFERENCE**

*FILE_POS* works exactly as *FPOS* but does not use a default channel. *FILE_PTRA* and *FILE_PTRR* move the file pointer, which may also be set with *GET*, *PUT*, *BGET* and *BPUT*.

## 13.68 FPOS_A

| | |
|---|---|
| Syntax | FPOS_A ([#ch,] pos) |
| Location | BTool |

This is the same as *FILE_PTRA*.

## 13.69 FPOS_R

| | |
|---|---|
| Syntax | FPOS_R ([#ch,] offset) |
| Location | BTool |

This is the same as *FILE_PTRR*.

## 13.70 FPUT$

| | |
|---|---|
| Syntax | FPUT$ [#ch,] string *[,string$^i$]* |
| Location | BTool |

This command writes the given string(s) in internal format to #ch, default is #1. The internal format of a string is a word (two bytes) giving the length of the string followed by the contents of the string itself.

**Example**

```
FPUT$ 'Hello'
```

will produce the equivalent of PRINT CHR$(0)&CHR$(5)&'Hello'.

**CROSS-REFERENCE**

*FGET$*, *PUT*.

## 13.71 FPUT%

| | |
|---|---|
| Syntax | FPUT% [#ch,] integer *[,integer$^i$]* |
| Location | BTool |

This command writes the specified integer(s) (range 0…32767) in its internal format to #ch, default is #1. An integer is stored internally as two bytes (one word).

**CROSS-REFERENCE**

*FGET%*, *PUT*

## 13.72 FPUTB

| | |
|---|---|
| Syntax | FPUTB [#ch,] {byte \| string$} *[,{byte$^i$ \| string$^i$$}]* |
| Location | BTool |

FPUTB is a command which writes single or multiple bytes to a channel #ch (default #1). FPUTB can take any kind of parameters which must be either a numeric value byte in the range 0..255 for a single byte, in the range 256..32767 for two bytes or a string string$.

**Example 1**

```
CLS: FPUTB "First line",10,"Second line"
CLS: FPUTB "First line",2570,"Third line"
```

because CVI% (CHR$ (10) & CHR$ (10) ) = 2570) which is (10 * 256) + 10 in big-endian format, as the QL is.

**Example 2**

FPUTB is very handy for controlling printers:

```
OPEN#3,par
FPUTB#3,27,"x",1
CLOSE#3
```

will enable near letter quality (NLQ) on an EPSON compatible printer.

**CROSS-REFERENCE**

*BPUT*, *CHR$*, *CODE*. *FGETB* is a complementary function.

## 13.73 FPUTF

| | |
|---|---|
| Syntax | FPUTF [#ch,] float *[,float$^i$]* |
| Location | BTool |

This command writes the floating point number(s) float in its internal format (six bytes) to #ch, default is #1.

**CROSS-REFERENCE**

*FGETF*, *PUT*

## 13.74 FPUTL

| | |
|---|---|
| Syntax | FPUTL [#ch,] longint *[,longint$^i$]* |
| Location | BTool |

This command writes the specified long integer(s) longint (-231..231-2) in internal format (four bytes) to #ch, default is #1.

**CROSS-REFERENCE**

*FGETL*, *PUT*

## 13.75 FRACT

| | |
|---|---|
| Syntax | FRACT (x) |
| Location | FRACT |

The function FRACT separates the fractional part of any floating point number x. It could easily be rewritten in SuperBASIC as the following:

```
100 DEFine FuNction MYFRACT (x)
110   RETurn x - INT(x) - (x < 0)
120 END DEFine MYFRACT
```

**CROSS-REFERENCE**

*TRINT* is complementary to *FRACT*.

## 13.76 FREAD

| | |
|---|---|
| Syntax | FREAD (#ch,address,bytes) |
| Location | TinyToolkit |

The function FREAD reads a number of bytes (bytes) from a given channel into memory, starting at address. The number returned by FREAD gives the number of bytes it actually read.

**Example**

A BASIC Procedure APPEND which adds a file (file1$) to the end of a target file (file2$). If the target file does not exist, it will be created.

The first file will be erased (remove line 220 if you do not want this). The third parameter allows you to determine the working space of the procedure; the larger this space, the quicker the execution:

```
100 DEFine PROCedure APPEND (file1$,file2$,bufsize)
110   LOCal length,buffer,file1,file2,part
120   file1=FOP_IN(file1$): length=FLEN(#file1)
130   buffer=ALCHP(length)
140   file2=FOPEN(file2$)
150   IF file2=-7 THEN file2=FOP_NEW(file2$)
160   GET #file2 \1E9
170   FOR part=0 TO INT(length/bufsize)
180     bufsize=FREAD(#file1,buffer,bufsize)
190     FWRITE #file2,buffer,bufsize
200   END FOR part
210   CLOSE #file1, #file2: RECHP buffer
220   DELETE file1$
230 END DEFine APPEND
```

It can be called as follows:

```
APPEND "ram1_tumb_tmp" TO "flp2_tump_dat",20480
```

**NOTE**

If the channel number supplied to FREAD does not refer to a file, then the error -15 (bad parameter) will be reported after it has done its work. This behaviour is pretty strange.

**CROSS-REFERENCE**

*FREAD$*, *FWRITE*, *LBYTES*, *SBYTES*, *GET*, *PUT*.

## 13.77 FREAD$

| | |
|---|---|
| Syntax | FREAD$ ([#ch], length) |
| Location | BTool |

The FREAD$ function is very similar to the FREAD command: A fixed number of characters is read from a channel (default #0) and returned as a string. FREAD$ does not stop with an error if the end of file is reached - you have to detect this by testing if the length of the returned string is really length.

**Example**

MYCOPY copies a file with flexible buffering up to 32k, eg. type:

```
MYCOPY "mycopy_bas" TO "ram2_whatever_dat", 1000
```

to use a 1000 bytes buffer. The larger the buffer, the faster the file is copied; try a one byte buffer to see the difference! Ok, here is the listing:

```
100 DEFine PROCedure MYCOPY (file1$, file2$, bufsiz%)
110   LOCal ch1, ch2, buffer$
120   ch1 = FOP_IN(file1$)
130   ch2 = FOP_NEW(file2$)
140   REPeat copying
150     buffer$ = FREAD$(#ch1, bufsiz%)
160     PRINT#ch2,buffer$;
170     IF LEN(buffer$) < bufsiz% THEN EXIT copying
180   END REPeat copying
190   CLOSE #ch1, #ch2
200 END DEFine MYCOPY
```

**CROSS-REFERENCE**

*INPUT$*, *FWRITE$*, *COPY*, *GET_BYTE$*

## 13.78 FREE

| | |
|---|---|
| Syntax | FREE |
| Location | BTool |

This function returns the largest block of the available free memory. This can be less than the actual free memory if the heap has become fragmented (see DEL_DEFB).

**CROSS-REFERENCE**

See also *FREE_MEM* and *TPFree*.

```
x=ALCHP(FREE)
```

reserves the largest piece of memory available.

## 13.79 FREE_FAST

| | |
|---|---|
| Syntax | FREE_FAST |
| Location | ATARI_REXT for QVME (v2.31+) |

The Atari TT recognises two types of RAM, standard ST RAM (up to 10MB) and FastRAM (otherwise known as TT RAM) which is specifically designed for the Atari TT and works about twice as fast as the standard ST RAM.

The QL emulator can use both types of RAM but will only recognise and use a maximum 4MB of standard ST RAM.

If FastRAM is available, the Emulator places the device drivers into this area in order to speed them up as well as freeing additional standard ST RAM. However, if your programs are to access the FastRAM, they need to use various new commands. FREE_FAST is a function which returns the amount of available FastRAM.

**CROSS-REFERENCE**

The other commands to access FastRAM are *RESFAST*, and *LRESFAST*.

Compare *RESPR*, *ALCHP* and *FREE_MEM*.

## 13.80 FREE_FMEM

| | |
|---|---|
| Syntax | FREE_FMEM |
| Location | SMSQ/E for Q68 |

Returns the amount of free fast memory on a Q68.

**Note 1**

Should be about 10 kB on a freshly booted Q68

**CROSS-REFERENCE**

*ALFM*

## 13.81 FREE_MEM

| | |
|---|---|
| Syntax | FREE_MEM |
| Location | Toolkit II, THOR XVI |

Exactly the same as *FREE*.

## 13.82 FREEZE

| | |
|---|---|
| Syntax | FREEZE switch (switch=ON or OFF) |
| Location | BTool |

The keys <CTRL><F5> cause the QL to stop working until any further key (except <CTRL>, <SHIFT>, <ALT> and <CAPSLOCK>), including <CTRL><F5>, is pressed, which will reactivate the QL.

This keystroke is generated by some commands to give the user a chance of reading the output, eg. VIEW, EXTRAS, SXTRAS, WDIR.

FREEZE OFF disables <CTRL><F5>, FREEZE ON re-activates it.

**Example**

```
FREEZE OFF
EXTRAS
FREEZE ON
```

## 13.83 FREEZE%

| | |
|---|---|
| Syntax | frozen = FREEZE% |
| Location | BTool |

FREEZE% returns either 0 or 1 (for OFF or ON respectively) if <CTRL><F5> has been disabled by FREEZE or not.

**Example**

```
frozen = FREEZE%
IF frozen THEN do_stuff: END IF
```

**CROSS-REFERENCE**

*ON* and *OFF* are constant expressions for 1 and 0. *FREEZE%* returns the current state. Compare *FREEZE* and *FREEZE%* to *BREAK* and *BREAK%*.

## 13.84 FSERVE

| | |
|---|---|
| Syntax | FSERVE or<br>FSERVE [device_name] (THOR XVI - v6.41 only) |
| Location | Toolkit II (hardware version only or SMS), THOR XVI |

This command creates a small fileserver job named Server which allows other network stations (slaves) to access all devices on the machine where this fileserver is running (this is the Master).

The fileserver only works with the QNET network system, which itself only works reliably if Toolkit II is installed as firmware (ie. on ROM or on EPROM) (or if Toolkit II is installed as part of SMS) on all machines connected to the network.

To access a device on the Master, a prefix has to be added to the device name. This prefix specifies the other machine by its network number (see NET) which may range from 1 to 8. The prefix consists of an n, the number of the remote station and an underscore, ie: n1_ .. n8_.

If an access fails for any reason, the sending machine will not receive an acknowledgement from the receiving one. In such cases, the network driver continues to try to get through for about 20 seconds and then reports 'Network aborted' (in #0) if it still cannot communicate with the specified machine.

**Examples**

```
OPEN#3,n3_scr: PRINT#3,"Bye.": FLUSH#3: CLOSE#3
WDIR n1_flp1_
```

```
FORMAT n7_win1_
```

Be careful with this sort of thing!

```
SAVE n2_ram1_PROGGY_bas
```

**NOTE 1**

All commands which use the SD.EXTOP or SD.FOUNT machine code calls will not work across the network: CHAR_USE for example. This does not necessarily mean that these commands report errors: CHAR_USE, for instance, changes the character set to a strange pattern. FORMAT will also fail over the Network. ED and EDIT also cannot be used to edit a program on a window opened over the Network.

**NOTE 2**

Although windows (scr_ and con_ devices) are normally not buffered, this will be the case if they are opened across the network. This affects just text output, all other operations (BORDER, INK, CLS, WINDOW etc.) are performed on the host QL when issued. The buffer of 256 bytes is located in the sending QL and flushed automatically if full. Otherwise a CLOSE command forces the buffer contents to be sent (the FLUSH command will not work to send the buffer contents). See the Drivers section in the Appendix for further details on Networks.

**NOTE 3**

If a channel was opened by a slave via the network and this QL is later removed from the network - say by unplugging the network lead or by resetting the machine, then the channel is left open. As all such channels are owned by the Server job, they can be flushed and closed by removing and restarting the job:

```
RJOB Server
FSERVE
```

Take care that all operations being carried out by other stations on the local machine (where the fileserver is to be removed) have finished or have been suspended.

**NOTE 4**

Due to checksum tests, data transmission across the network is practically error free. There is still a very small statistical possibility of transmission errors but really extensive experiments (moving megabytes of data) did not even produce one.

**NOTE 5**

Although a normal file name can be of any length up to a maximum of 41 characters (including the device name), if the file is to be accessed across the network, this is reduced to a maximum of 39 characters (including the network prefix). For example:

```
OPEN #3,flp1_Quill_letters_Minerva2_update_doc239
OPEN #3,n1_win1_Quill_letters_Minerva2b_updates
```

will work, whereas the following two commands report 'Not Found' without attempting to access the drives:

```
OPEN #4,flp1_Quill_letters_Minerva2_update_doc2392
OPEN #4,n1_win1_Quill_letters2_Minerva2b_updates
```

**NOTE 6**

If you OPEN a con_ device over the Network (onto a Master machine's screen) and try to use INPUT to read a variable entered on that Master, there are problems here in that the delete keys on the Master which is displaying (and editing) the text displayed in the con_ device do not work properly, leaving splodges on the screen. You can use IO_TRAP and QTRAP to call cursor positioning routines on the Master and then print spaces to overwrite the deleted characters, using IO_TRAP or QTRAP to move the cursor back to the correct position and possibly pan the window to get rid of excess characters. This technique was used to good effect in the NetPal program in DIY Toolkit (Vol N).

**NOTE 7**

If you try to use a Toolkit II command such as DIR to direct the output onto a window which has been OPENed over the Network, when it reaches the bottom of a page, the Toolkit II command automatically generates a <CTRL><F5> at the slave machine end which can only be cleared by pressing a key on the slave machine's keyboard.

**THOR XVI NOTE**

The THOR XVI version of this command allows you to send, for example, a continuous log of status messages to a file or device, eg. FSERVE scr_512x256a0x0. This is however really only useful for debugging network programs or to analyse network traffic.

**CROSS-REFERENCE**

The fileserver job can be removed with *RJOB*, *KJOB*, *KILL* etc. or by using a desktop application (such as QPAC2). See *NET* and *NFS_USE* for further information on networking.

Refer to the original documentation of Toolkit II and the Device Drivers Appendix for technical details. *SERNET* and *MIDINET* create fileservers for other Networks supported by SMSQ/E and the Atari ST

Emulators.

## 13.85 FSETH$

| | |
|---|---|
| Syntax | FSETH$ [#ch,] header$ |
| Location | BTool |

FSETH$ is a command which is the counterpart of FGETH$: it accepts either a 14 or 64 bytes long string which contains a file header (or at least the first part of that) and sends that file header to the specified channel (default #3).

**CROSS-REFERENCE**

See *HEADS* and *SetHEAD*!

## 13.86 FTEST

| | |
|---|---|
| Syntax | FTEST (name) |
| Location | Toolkit II |

The function FTEST is designed to allow you to test for the status of a file with the specified name. It will return a value of 0 if the given name can be opened for input only. It may however return a negative number representing an error code which would result if you tried to OPEN or OPEN_IN that file.

**NOTE 1**

The return of -6 (channel not open) has a special meaning in relation to this function, it means that the function could not find any room in the channel table to try and access the file.

**NOTE 2**

Due to the nature of the command, name can be used to represent any valid device, and could therefore, for example, be used to check if a resolution of 768x280 pixels is supported:

```
100 a$='scr_768x280a0x0'
110 IF FTEST(a$)<0
120   a$='scr_512x256a0x0'
130 END IF
140 OPEN #3,a$
```

**NOTE 3**

On Level-2 and Level-3 devices, there is always a file with the same name as the actual name of the device (eg. 'flp1_'). This therefore allows you to check if a medium is present in a Level-2 device:

```
IF FTEST(flp1_) < 0 THEN PRINT 'Please insert disk'
```

You must however be aware that on Level-1 devices, it is unlikely that such a file will be present and that FTEST will return -7 even if there is a disk present.

**NOTE 4**

FTEST will not warn you if a disk is read only, which can create problems.

**CROSS-REFERENCE**

*FOPEN* and *FOP_IN* allow you to open files safely. *DMEDIUM_RDONLY* can be used to find out if a disk is write protected.

## 13.87 FTYP

| | |
|---|---|
| Syntax | FTYP [(#channel)] or<br>FTYP (\file) (Toolkit II and THOR only) |
| Location | Toolkit II, THOR XVI, BTool |

This function returns the file type of a file which is already open as #channel (the default channel is #3) or else the second variant can be used (which supports the Toolkit II default data device and sub-directories) to check a given file.

The file type is one byte in the file header which by convention represents the type of the file. There are only four standard types:

- FTYP = 1 are executable jobs (normally suffixed _exe);
- FTYP = 2 are Sinclair Relocatable Object File (SROFF) modules (normally suffixed _REL);
- FTYP = 255 are sub-directories on level-2 and level-3 drivers;
- FTYP = 0 are everything else.

However, some programmers use their own file types for their applications, for example:

- FTYP = 2 may also signify sub-directory declaration files used by Ralf Biedermann's flp utility and Hirschbiegel drivers;
- FTYP = 3 are sub-directories on THOR computers.
- FTYP = 4represents font files used by the PAINTER.
- FTYP = 5 are pattern files used by the PAINTER.
- FTYP = 6 or 11 are compressed screens generated by the PAINTER.
- FTYP = 70 is used to represent separation files from packages distributed by the Intergroup Freeware Exchange.

**WARNING**

Sometimes machine code files (Toolkits, for instance) which should be loaded with LBYTES, LRESPR etc. have the file type 1. Executing such a file will lead to a crash in most cases, UNJOB changes the file type back to 0. Authors with a lot of skill write machine code which can be either executed as a job or loaded as a resident command executing the job from memory when called.

**CROSS-REFERENCE**

See *HEADR* for reading the whole file header and *EX* for executing jobs. *FILE_TYPE* does the same as *FTYP* but has a slightly different syntax.

## 13.88 FuNction

| | |
|---|---|
| Syntax | ... FuNction |
| Location | QL ROM |

This keyword forms part of the structure DEFine FuNction. As such, it cannot be used on its own within a program - this will cause a 'bad line' error.

**CROSS-REFERENCE**

Please refer to the individual structure descriptions for more details.

## 13.89 FUPDT

| | |
|---|---|
| Syntax | FUPDT [(#channel)] or<br>FUPDT (\file) (Toolkit II only) |
| Location | Toolkit II, BTool |

This function returns the date on which a given file was last amended. The value returned is the date in QDOS format, ie. the number of seconds since Midnight 1st January 1961. You can check this initial date with:

```
PRINT DATE$(0)
```

The update time is altered whenever a file is created or amended. A file which has overwritten a previous file or is a copy is regarded as a new file and will therefore have a different update time to the original. The default data device and sub-directories are supported, default channel is #3.

**Example**

It could be interesting to list all files which have been created during a certain time period. A simple prototype of a program which will do just that follows on below.

If you want to check all files, then dev$ should contain no sub-directories or wild cards (just FLP2_, WIN1_) and wild$ an empty string. Such a program could be used to write an intelligent backup program.

In order to run the program you will need TinyToolkit's TCONNECT or DIY-TK's QLINK. You could also use similar commands in the toolkits provided with Turbo or Qliberator.

```
100 CLS: INPUT "Device:"!dev$ \"Wild card:"!wild$
110 INPUT "List from (dd mm yy):"!first$
120 INPUT TO 2;"to (<ENTER>=today):"!last$\
130 day1=first$(1 TO 2):month1=first$(4 TO 5)
135 year1=19&first$(7 TO 8)
140 IF LEN(last$) THEN
150   day2=last$(1 TO 2):month2=last$(4 TO 5)
155   year2=19&last$(7 TO 8)
160 ELSE last=DATE
170 END IF
180 DATE_tmp=DATE
190 SDATE year1,month1,day1,0,0,0: first=DATE
```

(continues on next page)

(continued from previous page)

```
200 IF LEN(last$): SDATE year2,month2,day2,23,59,58: last=DATE
210 ADATE DATE_tmp-DATE+2
220 :
230 OPEN#3,pipe_10000: OPEN#4,pipe_100
240 TCONNECT #3 TO #4: WDIR#3,dev$ & wild$
250 yes=0: yesno=0
260 REPeat show_those
270   IF NOT PEND(#4) THEN EXIT show_those
280   INPUT#4,file$: this=FUPDT(\dev$ & file$): yesno=yesno+1
290   IF first<=this AND this<=last THEN
300     PRINT file$;TO 20;"(";DATE$(this);")"
310     yes=yes+1
320   END IF
330 END REPeat show_those
340 PRINT \"(";yes;"/";yesno!"files)"
350 CLOSE#3,#4
```

Minerva or SMS users can delete lines 180,190,200 and 210 and use the following lines instead:

```
190 first=DATE(year1,month1,day1,0,0,0)
200 IF LEN(last$): last=DATE(year2,month2,day2,23,59,58)
```

**NOTE 1**

The update time of a file will only be correct if the system clock was set to the correct time when the file was last written to, since it is the date contained within the QL's clock which is written to the header of the file. If your machine has a battery backed real-time clock, then this presents no real problem; otherwise you will need to ensure that you set the date and time after each startup.

**NOTE 2**

On some early versions of Toolkit II the machine code FS.RENAME routine also alters the update time of a file!

**NOTE 3**

There is no legitimate way to change the update time of a file except with level-2 or level-3 drivers. It is of course possible to set the system clock temporarily to the desired time, amend the file and then set the clock back, but as the time taken to change the cannot be estimated exactly, this method will almost surely reset the system clock to the incorrect time after carrying out such an operation a few times. On level-2 and level-3 drivers, SET_FUPDT can be used.

**NOTE 4**

Minerva automatically updates the update dates of files on microdrives. Other ROM versions will not do so without Toolkit II.

**CROSS-REFERENCE**

*FBKDT*, *FLEN*, *FTYP*, *FDAT*, *FXTRA*, *FILE_LEN*, *FILE_TYPE*, *FVERS* and *FNAME$* return other information about a file. See *DATE* and *ADATE* about handling the system clock and *SET_FUPDT* on setting the time stamp of a file.

## 13.90 FVERS

| | |
|---|---|
| Syntax | FVERS [(#channel)] or<br>FVERS (\file) |
| Location | Level-2 Device Drivers |

This function reads the version number of the given file (or of the file attached to the specified channel {default #3} if the first variant is used).

The version numbers can range from 0 to $2^{16}$-1 (65535) and generally indicate how often a file has been amended. If a file was created on a level-1 device driver system, its version number is zero (0), while newly created files on level-2 device drivers will have the version number 1 after they have been closed. FVERS supports Toolkit II's default data device and sub-directories.

If the first variant is used, the default channel is #3 if none is specified.

Each time that a file is amended on level-2 and level-3 drivers, the version number is increased by one. If the version reaches its limit of 65535, it will start at version 1 again. A file has to be re-opened to change its version by more than one. After the file has been amended, the version will only increase after a FLUSH or CLOSE.

Unfortunately current versions of SAVE and SBYTES do not increase the version number because they overwrite existing files instead of truncating them.

**Example**

```
OPEN_OVER#3,test_tmp: REMark create the file
PRINT#3,"just a line": REMark write a line to the file
PRINT FVERS(#3): REMark 0, neither flushed nor closed
CLOSE#3: REMark close file
PRINT FVERS(\test_tmp): REMark 1
OPEN#3,test_tmp: REMark re-open file
PRINT FVERS(#3): REMark 1, nothing changed yet
PRINT#3,"replace the line": REMark amend file
PRINT FVERS(#3): REMark still 1 not yet flushed
FLUSH#3: REMark write slave blocks to file
PRINT FVERS(#3): REMark now it's 2
FLUSH#3: REMark flush again
PRINT FVERS(#3): REMark  2
PRINT#3,"next line": REMark change file again
CLOSE#3: REMark close file
PRINT FVERS(\test_tmp): REMark  still 2
```

**NOTE**

The file version number is not preserved if the file is overwritten. However, if you make a copy of a file, this keeps the same version number as the original, but not on Level-1 drivers.

**SMS NOTE**

If you use the SAVE command without a filename to save a previously loaded SuperBASIC program, the file version number will be increased by one.

**CROSS-REFERENCE**

*SET_FVERS* allows you to set the version number.

## 13.91 FWRITE

| | |
|---|---|
| Syntax | FWRITE #ch,address,bytes |
| Location | TinyToolkit |

The command FWRITE reads a given number of bytes (bytes) from memory (starting at address) and writes them to the given channel, which should point to a file.

**Example**

A procedure which adds a file to another already existing file - CONCAT cannot do so. This is a rather primitive version which grabs as much memory as necessary and uses only TinyToolkit extensions. The program is very primitive (not in its use of these extensions), but because the memory management of the routine is simple (but fast) and as FILE_LEN does not support default devices and sub-directories, full filenames have to be passed.

See *FREAD* for an enhanced version!

```
100 DEFine PROCedure APPEND (file1$,file2$)
110   length=FILE_LEN(file1$)
120   buffer=GRAB(length)
130   LBYTES file1$,buffer
140   ch=FOPEN(file1$): FILE_PTRA#3,1E9
150   FWRITE #ch,buffer,length
160   CLOSE#ch: RELEASE buffer
170 END DEFine APPEND
```

The procedure is called by APPEND file1$ TO file2$, which will add the first file to the second file. First, a buffer of the size of the first file is reserved in RAM, then, this file is read into the buffer. Now the second file is opened, the file pointer moved to the end of it and the whole buffer is then appended to the end of the file. Finally, the channel is closed and the buffer RELEASEd.

**NOTE**

If the channel number does not refer to a file or even if the channel has not yet been opened, FWRITE will report error -15 (bad parameter) after it has completed its work. This behaviour is pretty strange.

**CROSS-REFERENCE**

*FREAD*, *LBYTES*, *SBYTES*, *GET*, *PUT*. If you intend to use *APPEND*, please see *FREAD* for a better version.

## 13.92 FWRITE$

| | |
|---|---|
| Syntax | FWRITE$ [#ch,] string$ |
| Location | BTool |

FWRITE$ is a command (not a function as the $ may suggest) and writes string$ to #ch (default #1). It's the same as PRINT#ch,string$;.

**Example**

```
FWRITE$ "Hello World"
```

**NOTE**

The Line feed character {CHR$(10)} is *not* printed at the end of the text.

## 13.93 FXTRA

| | |
|---|---|
| Syntax | FXTRA [(#ch)] or<br>FXTRA \file (Toolkit II only) |
| Location | Toolkit II, BTool |

This is a function which returns part of the file header relating to the specified file (or the file attached to the specified channel {default #3} if the first variant is used). See FGETH$ for what part of the file header FXTRA returns. The Toolkit II default data device and sub-directories are supported. If the first variant is used, the default channel is #3.

**CROSS-REFERENCE**

See *FDAT*, *FBKDT*, *FUPDT* and *FTYP* which return similar information.

CHAPTER

# FOURTEEN

# KEYWORDS G

## 14.1 GCD

| | |
|---|---|
| Syntax | GCD ($x^1$, $x^2$ *[,$x^i$]*) Where $x^i$=0..INTMAX |
| Location | Math Package |

The function GCD takes two or more positive integers and finds their greatest common denominator, ie. the largest number by which all of the given parameters can be divided without remainder. There is always such a number because 1 (the smallest common denominator) divides every number without remainder. GCD converts all passed values into integers by removing any decimal places (as with *INT*) before looking for the denominator.

**CROSS-REFERENCE**

*LCM*

## 14.2 GER_MSG

| | |
|---|---|
| Syntax | GER_MSG |
| Location | ST/QL |

The file GER_TRA_rext is supplied with the ST/QL Emulator which contains translation tables to allow the Emulator to use German. Once this file has been LRESPR'd, this function can be used to find the start of the message translation table to be used with the TRA command. You can use: TRA GER_TRA,GER_MSG to set up the printer and message translation tables for Germany.

**CROSS-REFERENCE**

See *NOR_MSG* and *GER_TRA*. Also see *TRA*.

## 14.3 GER_TRA

| | |
|---|---|
| Syntax | GER_TRA |
| Location | ST/QL |

This is the complementary function to GER_MSG and points to the printer translation table for Germany contained in the file GER_TRA_rext.

**CROSS-REFERENCE**

See *GER_MSG*.

## 14.4 GET

| | |
|---|---|
| Syntax | GET [#channel\file_position,] [var[1] *[,var[i]]* ...] or<br>GET [#channel,] [var[1] *[,var[i]]* ...] |
| Location | Toolkit II, THOR XVI |

This command, together with PUT, helps to provide the QL with a means of using a file as a store for variables. The QL stores variables in one of four ways: short%(range -128 to 127) is stored as 2 bytes. Short integers are only available on Minerva ROMs, when integer tokenisation is enabled. integer%(range -32768 to 32767) is stored as 4 bytes. float(eg. 1.23 or any numbers outside the integer% range) is stored as 6 bytes. string$(eg. 'Hello') is stored as 2 bytes containing the length of the string followed by the string itself.

GET enables variables which have been stored in this manner to be retrieved from a file opened to the given channel (default #3). The variable stored at the current position in the file (or the file position given with the command, if the first variant is used) is read directly into the variable name given with the command.

If you provide more than one variable name as the second, third parameter etc, then more variables will be read from the file in one go.

If the first variant is used, the file position is always calculated as an absolute distance from the start of the file. However, to help you, if you supply a variable for the file_position (eg. GET \pointer), this variable will always be updated to the current file pointer at the end of the command.

Compare GET \pointer+3 which supplies an expression for the file_pointer and cannot therefore be updated automatically by the command.

If no variable is specified, the file pointer will be set to the specified position if the first variant is used.

If the second variant is used, this will have no effect.

**Example**

A program to store three names on a file and then to retrieve them (this would be useful in a database for instance):

```
100 RESTORE
110 DIM a$(3,20)
120 FOR i=1 TO 3: READ a$(i)
130 OPEN_NEW #3,ram1_storage
140 PUT #3,a$(1),a$(2),a$(3)
150 CLOSE #3 160 :
170 REPeat opt_loop
180   CLS: INPUT 'Which Number Name do you want?',no$
190   IF no$='':NEXT opt_loop
200   IF no$(1) INSTR '123': opt=no$(1): EXIT opt_loop
210 END REPeat opt_loop
220 OPEN_IN#3, ram1_storage
```

(continues on next page)

(continued from previous page)

```
230 FOR i=1 TO opt:GET#3,retrieve$
240 PRINT retrieve$
250 CLOSE #3
260 DATA 'Fred Bloggs','Filthy Rich','Peter Rabbit'
```

**NOTE 1**

The example works fine if only a few fields have to be stored. Generally, it is better to move around a file using file pointers in a file based database.

**NOTE 2**

Current versions of the Turbo and Supercharge compilers are not able to compile programs which use GET.

**NOTE 3**

Except under SMS v2.81+, this command can crash the system if you try to GET a string variable which has been dimensioned {or even set with LOCal a$(512) for example}. This can be avoided by using:

```
a$=FILL$(' ',512)
```

to initialise the string instead.

**NOTE 4**

Although it is possible to use this command with non-file related channels, this is inadvisable, as each entry would need to be typed in from the keyboard in its internal form, which can be rather difficult. If you do use the command on a non-file related channel by accident, press the Break key to escape.

**CROSS-REFERENCE**

See *PUT*, *BPUT*, *BGET*, *LGET*, *WGET*.

## 14.5 GET_BYTE$

| | |
|---|---|
| Syntax | GET_BYTE$ (#channel,bytes) |
| Location | TinyToolkit |

This function will read a specific number of bytes from the given channel and return the result as a string. If GET_BYTE$ cannot get the specified number of bytes from that channel, it will wait until there are enough bytes present in the channel or until it detects an End Of File character. GET_BYTE$ does not care which characters are read, so <LF> = CHR$(10) will not cause any problems unlike INPUT.

**Example**

A program to compare the contents of two files, both of which are the same length. The greater the buffer size (maximum 32767 bytes), the faster will be the execution, but then again the greater the work space which will be needed (maximum 64K). This is an example of the fundamental link between available memory and operation speed:

```
100 File1$="ram1_a"
110 File2$="ram1_b"
120 Buffer=10000
```

(continues on next page)

(continued from previous page)

```
130 :
140 Pieces=FILE_LEN(File1$) DIV Buffer
150 Rest=FILE_LEN(File1$) MOD Buffer
160 OPEN#3,File1$: OPEN#4,File2$
170 FOR Blk=0 TO Pieces+1
180   IF Blk>Pieces THEN Buffer=Rest
190   One$=GET_BYTE$(#3,Buffer)
200   Two$=GET_BYTE$(#4,Buffer)
210   PRINT "Block"!Blk TO 12;
220   IF One$<>Two$ THEN
230     PRINT "Difference between"!Buffer*Blk!"and"! Buffer*(Blk+1)
240   ELSE
250     PRINT "OK"
260   END IF
270 END FOR Blk
280 CLOSE#3: CLOSE#4
```

**NOTE**

Earlier TinyToolkit versions (pre v1.10) called this function GET$, which unfortunately caused problems with a similar function in the Turbo Toolkit and EASYPTR.

**CROSS-REFERENCE**

*INKEY$* reads just one byte from the given channel, which is therefore much slower than *GET_BYTE$* if blocks of bytes are to be read. On the other hand, *INKEY$* allows you to specify a timeout.

The *INPUT* command combines input/output and reads blocks, but a block must end with <LF>.

The usage of the different keywords depends mainly on the structure of the incoming data. User input and lines in an ASCII file normally terminate with Enter <LF>, while internal data such as disk directory entries are stored as blocks with a fixed length (see *FOP_DIR*). Have a look at *GET*, *PUT*, *BGET* and *BPUT*, too. *FILE_PTRR*, *FILE_POS*, *FPOS* can be used for movement.

## 14.6 GET_BYTE

| | |
|---|---|
| Syntax | byte = GET_BYTE(#channel) |
| Location | DJToolkit 1.16 |

Reads one character from the file attached to the channel number given and returns it as a value between 0 and 255. This is equivalent to CODE(INKEY$(#channel)).

BEWARE, *PUT_BYTE* can put negative values to file, for example -1 is put as 255, GET_BYTE will return 255 instead of -1. Any negative numbers returned are always error codes.

**EXAMPLE**

```
c = GET_BYTE(#3)
```

**CROSS-REFERENCE**

*GET_FLOAT*, *GET_LONG*, *GET_STRING*, *GET_WORD*.

## 14.7 GET_FLOAT

| | |
|---|---|
| Syntax | float = GET_FLOAT(#channel) |
| Location | DJToolkit 1.16 |

Reads 6 bytes from the file and returns them as a floating point value.

BEWARE, if any errors occur, the value returned will be a negative QDOS error code. As GET_FLOAT does return negative values, it is difficult to determine whether that returned value is an error code or not. If the returned value is -10, for example, it could actually mean End Of File, this is about the only error code that can be (relatively) safely tested for.

**EXAMPLE**

```
fp = GET_FLOAT(#3)
```

**CROSS-REFERENCE**

*GET_BYTE*, *GET_LONG*, *GET_STRING*, *GET_WORD*.

## 14.8 GET_LONG

| | |
|---|---|
| Syntax | long = GET_LONG(#channel) |
| Location | DJToolkit 1.16 |

Read the next 4 bytes from the file and return them as a number between 0 and 2^32 -1 (4,294,967,295 or HEX FFFFFFFF unsigned).

BEWARE, the same problem with negatives & error codes applies here as well as *GET_FLOAT*.

**EXAMPLE**

```
lv = GET_LONG(#3)
```

**CROSS-REFERENCE**

*GET_BYTE*, *GET_FLOAT*, *GET_STRING*, *GET_WORD*.

## 14.9 GET_STRING

| | |
|---|---|
| Syntax | a$ = GET_STRING(#channel) |
| Location | DJToolkit 1.16 |

Read the next 2 bytes from the file and assuming them to be a QDOS string's length, read that many characters into a$. The two bytes holding the string's length are NOT returned in a$, only the data bytes.

The subtle difference between this function and *FETCH_BYTES* is that this one finds out how many bytes to return from the channel given, *FETCH_BYTES* needs to be told how many to return by the user. GET_STRING is the same as:

```
FETCH_BYTES(#channel, GET_WORD(#channel))
```

WARNING - JM and AH ROMS will give a 'Buffer overflow' error if the length of the returned string is more than 128 bytes. This is a fault in QDOS, not DJToolkit. The demos file, supplied with DJToolkit, has a 'fix' for this problem.

**EXAMPLE**

```
b$ = GET_STRING(#3)
```

**CROSS-REFERENCE**

*GET_BYTE*, *GET_FLOAT*, *GET_LONG*, *GET_WORD*, *FETCH_BYTES*.

## 14.10 GET_STUFF$

| | |
|---|---|
| Syntax | GET_STUFF$ |
| Location | GETSTUFF |

The Hotkey System II uses the keys <ALT><SPACE> and <ALT><SHIFT><SPACE> to type into the current keyboard buffer the contents of a certain piece of memory, known as the Hotkey Stuffer Buffer. The command HOT_STUFF text$ puts text$ into this buffer.

The function GET_STUFF$ returns the contents of the hotkey stuffer or "0" if it does not contain anything. If the FILES Thing of QPAC2 is present, this will be started first, prior to returning the stuffer contents. This means that a program can easily ask for a filename - just by calling GET_STUFF$.

**NOTE**

GET_STUFF$ returns cryptic numbers in unusual circumstances, for example:

```
HOT_STUFF ""
PRINT GET_STUFF$
```

**WARNING**

This function crashes SMSQ/E and Minerva when you Quit the Files Menu of QPAC 2.

**CROSS-REFERENCE**

See *HOT_STUFF*.

## 14.11 GetHEAD

| | |
|---|---|
| Syntax | GetHEAD #ch, adr |
| Location | HEADER (DIY Toolkit) |

GetHEAD loads the header of an opened file pointed to by the channel #ch into memory at adr, which must point to at least 64 bytes of reserved memory.

**Example**

If the file header of an executable file is lost then you must modify it so that the file can be executed again. Executable files need the file type set to 1 and the dataspace to be specified, the latter must be large enough to avoid a serious crash. MAKEJOB does this with file$, demonstrating GetHEAD and SetHEAD:

```
100 DEFine PROCedure MAKEJOB (file$, dataspace)
110 LOCal fp
120 fp=FOPEN(file$): IF fp<0 THEN STOP
130 adr=ALCHP(64): IF adr=0 THEN STOP
140 GetHEAD#fp,adr
150 POKE adr+5,1
160 POKE_L adr+6,dataspace
170 SetHEAD#fp,adr
180 CLOSE#fp: RECHP adr
190 END DEFine MAKEJOB
```

**CROSS-REFERENCE**

*SetHEAD* saves a file header. See *FGETH$* for information about the file header. *HEADR* is very similar to *GetHEAD*. See also *HGET* and *HPUT*.

## 14.12 GET_WORD

| | |
|---|---|
| Syntax | word = GET_WORD(#channel) |
| Location | DJToolkit 1.16 |

The next two bytes are read from the appropriate file and returned as an integer value. This is equivalent to CODE(INKEY$(#channel)) * 256 + CODE(INKEY$(#channel)). See the caution above for *GET_BYTE* as it applies here as well. Any negative numbers returned will always be an error code.

**EXAMPLE**

```
w = GET_WORD(#3)
```

**CROSS-REFERENCE**

*GET_BYTE*, *GET_FLOAT*, *GET_LONG*, *GET_STRING*.

## 14.13 GETXY

| | |
|---|---|
| Syntax | GETXY x%, y% |
| Location | HCO |

This command draws a crosshair (with its centre at (x%,y%) which can be moved with the cursor keys. Holding down <SHIFT> while pressing a cursor key will speed up movement. Once the crosshair is placed in the correct position, press <SPACE> to return to BASIC. The two parameters x% and y% will be updated to the position of the centre of the cross.

**NOTE 1**

It is obligatory to pass integer variables to GETXY.

**NOTE 2**

GETXY returns a wrong value for y% on Minerva ROMs, so it is unusable.

**NOTE 3**

Turbo and Supercharge compilers cannot compile this command.

**WARNINGS**

See *SET*.

**CROSS-REFERENCE**

*INVXY*

## 14.14 GO SUB

| | |
|---|---|
| Syntax | GO SUB line_number (GOSUB is expanded to GO SUB) |
| Location | QL ROM |

The command GO SUB was only implemented to make SuperBASIC more compatible with other versions of BASIC.

SuperBASIC offers much more elegant and powerful alternatives to this command - 'structured programming'. Structured programs do not have to be longer than the same program using GO SUB commands.

It is strongly recommended that you do not use GO SUBs in programs. A similar effect (and much more besides) can be achieved by using DEFine PROCedure and DEFine FuNction.

The idea behind GO SUB is that it jumps to a sub-routine within a program which starts at the specified line_number. Program flow then continues through that sub-routine until a RETurn statement is found, in which case, control is then returned to the statement following the original GO SUB.

**Example**

A simple program which prints a title in shadow writing, using GO SUB to call up the shadow writing routine:

```
100 MODE 8
110 WINDOW 448,200,32,16:PAPER 0:CLS
120 a$='Hello there World'
130 GO SUB 1000
140 PAUSE
150 CLS
160 :
999 STOP
1000 CSIZE 2,0
1010 AT 10,10:INK 4:PRINT a$
1020 CURSOR 42,56,10,10:INK 7:OVER 1:PRINT a$:OVER 0
1030 RETurn
```

This is actually much easier to read (and more flexible) if re-written to use DEFine PROCedure instead (note that there is no longer any need for line 999).

```
100 MODE 8
110 WINDOW 448,200,32,16:PAPER 0:CLS
130 SHADOW_PRINT "Hello there World'
140 PAUSE
150 CLS
160 :
1000 DEFine PROCedure SHADOW_PRINT(v$)
1010   CSIZE 2,0
1020   AT 10,10:INK 4:PRINT v$
1030   CURSOR 42,56,10,10:INK 7:OVER 1:PRINT a$:OVER 0
1040 END DEFine
```

**NOTE 1**

It is not a crime to use GO SUB in your programs, after all, machines are built for human beings, so the machines should be adapted to users, and users must all find the most comfortable way for them to use their machines.

**NOTE 2**

A calculated GO SUB statement, eg:

```
GO SUB 1000+x*100
```

although allowed by the interpreter, is unlikely to be compiled successfully. Secondly, RENUM is unable to change the line number of such GO SUBs. There were also problems with using an expression for GO SUB in SMS pre v2.59.

**NOTE 3**

Avoid using GO SUB in an in-line FOR loop - see Note 2 of FOR.

**CROSS-REFERENCE**

Try to use SuperBASIC's more powerful *REPeat*, *FOR*, *DEFine PROCedure* and *DEFine FuNction* structures instead!

## 14.15 GO TO

| | |
|---|---|
| Syntax | GO TO line_number(GOTO is expanded to GO TO) |
| Location | QL ROM |

The command GO TO behaves in a similar way to GO SUB in that it forces program flow to jump to a different part of the program. It is not possible to RETurn to the statement following GO TO, unless you use another GO TO command. SuperBASIC allows much more elegant and powerful structures which should be used.

**Example**

An extremely simple password check:

```
  10 INPUT Password$
  20 IF Password$=='QL lives' THEN GO TO 50
```

(continues on next page)

(continued from previous page)

```
   30 PRINT 'Access DENIED'
   40 GO TO 10
   50 PRINT 'Access Granted'

This would be much better if re-written::

   10 REPeat Pass_loop
   20   INPUT Password$
   30   IF Password$=='QL lives' THEN EXIT Pass_loop
   40   PRINT 'Access DENIED'
   50 END REPeat Pass_loop
   60 PRINT 'Access Granted'
```

**CROSS-REFERENCE**

Please read *GO SUB* before you dare to try *GO TO*!

## 14.16 GPOINT

| | |
|---|---|
| Syntax | GPOINT [#ch,] x,y [,$x^2$,$y^2$ [,$x^3$,$y^3$, . . . ] ] |
| Location | GPOINT |

This command is the same as *POINT* but fixes the bug in MGx ROMs.

## 14.17 GRAB

| | |
|---|---|
| Syntax | GRAB (bytes) |
| Location | TinyToolkit |

GRAB is a function which reserves a specified amount of space in the common heap area of memory for use and returns the start address of the allocated area.

**CROSS-REFERENCE**

With *GRAB* (unlike *ALCHP*), reserved memory can only be given back to QDOS for other purposes with *RELEASE*. It is necessary to know the start address returned by *GRAB* to do this, so a formula like *SCRBASE* GRAB(32768) wastes 32k of RAM if *SCRBASE* is used again. Although *GRAB* is comparable to *RESPR* in this respect, it will work with jobs in memory just like *ALCHP*. See also *RESERVE*. The amount of available memory can be found by using *FREE* or *FREE_MEM*.

## 14.18 GREGOR

| | |
|---|---|
| Syntax | GREGOR (day%, month%, year%) |
| Location | Math Package |

The function GREGOR takes three integers (floats & longs are rounded to the nearest integer) to specify a date and returns the weekday as a number from 1 to 7 where:

- 1 = Sunday, ( See Note 1 !)
- 2 = Monday,
- 3 = Tuesday,
- 4 = Wednesday,
- 5 = Thursday,
- 6 = Friday,
- 7 = Saturday.

As the name of the function suggests GREGOR uses the Gregorian calender.

This was introduced in 1583, so GREGOR has to refuse earlier years. Invalid parameters are not reported by breaking with an error (unless one of the parameters is out of integer range) but by returning zero.

**Example**

Print your own calendar!

```
100 CLS
110 REPeat getmonth
120   INPUT "Year:"!year;TO 12;"Month:"!month
130   firstday$ = GREGOR(1,month,year)
135   firstday=firstday$(1)
140   IF NOT firstday THEN
150     PRINT "Invalid input."
160   ELSE EXIT getmonth
170   END IF
180 END REPeat getmonth
190 FOR lastday = 28 TO 31
200   IF NOT GREGOR(lastday+1,month,year): EXIT lastday
210 END FOR lastday
220 :
230 PRINT \" Sun Mon Tue Wed Thu Fri Sat"
240 PRINT FILL$(" ",4*(firstday-1));
250 FOR day = 1 TO lastday
260   PRINT FILL$(" ",4-LEN(day));day;
265   xday$=GREGOR(day,month,year)
270   IF xday$(1) = 7 THEN PRINT
280 END FOR day
```

**NOTE 1**

GREGOR was originally intended to return 1 for Monday, 2 for Tuesday and so on. The current version (v2.05) follows the Christian tradition where Sunday was regarded as the first day of the week. The programming example above corrects this by applying this interpretation and uses:

```
230 PRINT " Sun Mon Tue Wed Thu Fri Sat"
```

instead of:

```
230 PRINT " Mon Tue Wed Thu Fri Sat Sun".
```

**NOTE 2**

Current versions (v2.05) of this command include a bug which mean that it will not work correctly on Minerva, SMSQ/E and possibly other ROMs.

**CROSS-REFERENCE**

*EASTER*, *DAY$*

## 14.19 GT$

| | |
|---|---|
| Syntax | GT$ (type, string1$, string2$) |
| Location | Btool |

This function allows you to compare two strings using the comparison types supported by QDOS - it is therefore more flexible than direct comparison using operators (see Appendix 11). The function will always return 1 if string1$ is greater than string2$ and is therefore similar to:

```
PRINT string1$ > string2$
```

However, you can specify one of four comparison types, which will affect the outcome:

| TYPE | Effect |
|---|---|
| 0 | Compare the two strings character by character |
| 1 | Ignore the case of the letters |
| 2 | If there is no difference in the characters, compare the values of any embedded numbers. |
| 3 | Ignore the case of the letters and still if there is no difference in the characters, compare the values of any embedded numbers. |

The characters are compared by using the following order:

```
SPACE
!"#$%&'()\*+,-/:;<=>?@[\]^_£{\|}~© 01234567890
AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
Foreign characters (in order of the character set)
```

**CROSS-REFERENCE**

See *INSTR*. GE, LT, LE, *EQ$* and NE are all similar facilities.

CHAPTER

FIFTEEN

# KEYWORDS H

## 15.1 HEADR

| | |
|---|---|
| Syntax | HEADR file, adr |
| Location | TinyToolkit |

This command reads the file header of the specified file and stores it at the specified address (adr) in memory. Note that file must be the full filename.

**CROSS-REFERENCE**

See *GetHEAD* (and *FGETH$*)!

## 15.2 HEADS

| | |
|---|---|
| Syntax | HEADS file, adr |
| Location | TinyToolkit |

This command sets the file header of the specified file to the 64 bytes stored at adr.

**CROSS-REFERENCE**

See *HEADR* and *SetHEAD* (and *FGETH$*)!

## 15.3 HEX

| | |
|---|---|
| Syntax | HEX (hex$) |
| Location | Toolkit II, THOR XVI |

This function returns the decimal value of a hexadecimal number. The hexadecimal system of numbering is based upon base 16, which means that instead of each digit being in the range 0..9 (as in decimal), each digit can actually hold the value 0..15.

Because a digit can only be one character, a system was devised for representing the value 10..15 - these values are represented by the letters A..F respectively. Any digits outside the range 0..F will cause an 'error in expression'.

Hex$ can be anything in the range $80000000 (which equates to -2^31) to $79999999 (which equates to 2^31-1).

**Examples**

```
PRINT HEX('F')
```

will print the value 15.

```
PRINT HEX('10')
```

will print the value 16.

**CROSS-REFERENCE**

*HEX$* works the other way around, converting decimal numbers into their hexadecimal equivalent. See *BIN* and *BIN$* for the binary versions. SMS users can achieve the same thing by using, for example:

```
PRINT $1AB
```

instead of:

```
PRINT HEX ('1AB').
```

## 15.4 HEX$

| | |
|---|---|
| Syntax | HEX$ (decimal,nibbles) or<br>HEX$ (decimal [,nibbles]) (THOR only) |
| Location | Toolkit II, THOR XVI |

This function converts a signed integer decimal number to the equivalent hexadecimal number to a specified number of nibbles (ranging from 1 to 32 - there are 4 nibbles in one digit). If nibbles is not divisible by four, it is rounded up to the next multiple of four. Negative values are also handled correctly provided that nibbles is set to 32. The range of decimal is -2^31<decimal<2^31

**Examples**

```
HEX (HEX$ (x,4))
```

Will = x if x is any number between 0 and 15.

```
PRINT HEX$ (300,10)
```

will return the value 12C.

```
PRINT HEX$ (300,8)
```

will return the value 2C.

**THOR XVI NOTE**

The THOR XVI version of this command will accept a value of zero for nibbles {or even the command in the form HEX$(decimal)}. In both of these cases the result is returned in the least number of Hexadecimal digits necessary to store the number, for example:

```
PRINT HEX$(32)
```

gives the result 20.

**THOR XVI WARNING**

A second parameter of zero may crash some versions of this command except on v6.41 of the THOR XVI.

**CROSS-REFERENCE**

See *HEX* and *BIN*, *BIN$*.

## 15.5 HGET

| | |
|---|---|
| Syntax | HGET [#ch] length [ , access [, type [, dataspace [, extra ]]]] |
| Location | SMSQ/E |

This command allows you to access the various parameters which are contained in the header of the file attached to the specified channel (default #3). The command will set the supplied parameters to the required information.

If the specified channel is not open to a file, then an Invalid Parameter error is reported.

The information returned is as follows:

- length = Length of the File
- access = File Access Key
- type = File Type
- dataspace = Dataspace for Executable Files
- extra = Extra Information

**NOTE**

You will not be able to compile this command with Turbo or SuperCharge due to the fact that it alters its own parameters.

**CROSS-REFERENCE**

*HPUT* saves the file header information. See *FGETH$* for information about the file header. See also *HEADR* and *GetHEAD*.

## 15.6 HIS_SET

| | |
|---|---|
| Syntax | HIS_SET #channel |
| Location | History Device (Needs Pointer Environment) |

The History Device is an extension to the internal QDOS routine IO.EDLIN which reads editable text input from the keyboard; IO.EDLIN is used if, for example, you use INPUT on a window - even the interpreter uses IO.EDLIN to get its commands from #0. But let's see how the History Device alters keyboard input.

The toolkit must be loaded into Resident Procedure Memory (with RESPR) before any Jobs are started, but after the Pointer Environment and Lightning / Speedscreen (or any other drivers which alter the con_ device driver) are installed (if either of these two packages are required).

Often, if you respond to a set of questions asked by a program, the same answers crop up again and again. For example, take the SuperBASIC interpreter, if you have entered a command, it will carry it out and wait for the next command to be entered. You may find that you want to type in the same text - or something which you entered a few loops ago again.

Toolkit II and the Hotkey System install a last line recall when the <ALT><ENTER> key combination is pressed (can be configured with the Hotkey System) - this is widely accepted (it's even supported by keyboard interfaces), but, has not proved to be very reliable or useful: recalling the last line typed generally works well, but for lines which were entered some time ago, things become easily messed up.

The History Device sits on the afore-mentioned QDOS trap and stores a specified number of entered lines for any console channel. If a program then requests input, previously entered lines can be recalled using the <↑> and <↓> keys - this is called a command line history (hence the name of the History Device).

A history for a console channel must be explicitly installed, by using the HIS_SET command. HIS_SET can only accept channel numbers which point to a con_ device, it breaks with 'bad parameter' if that is not the case. Once the history is installed, you will need to activate it by using the command HIS_SIZE.

**Example**

To install a command line history for the SuperBASIC interpreter - these lines could be added to a BOOT program:

```
HIS_SET #0: REMark Install a command line history for #0
HIS_SIZE #0, 40: REMark Activate history which stores 40 lines
```

**NOTE 1**

The History Device does not run on at least pre E.21 drivers of the ST/QL or STE/QL due to a bug in the IO.EDLIN trap of these drivers. History is fine for Minerva up to v1.93, although there are harmless problems with v1.96 in that the current line is not displayed before the line is altered. QView and Jochen Merz Software have been informed about these difficulties, so these may already be fixed by now.

**NOTE 2**

You cannot redefine the keys used for last line recall!!

**SMS NOTE**

The History device is built into SMSQ/E, although the HIS_... commands are not included with SMSQ/E, so in reality, you can only use the HISTORY device as a Last In First Out pipe system - see Devices Appendix. In any event the HIS_... commands do not appear to work on SMS.

**CROSS-REFERENCE**

A history is automatically removed when a channel is closed but removal can also be forced with *HIS_UNSET*. The QDOS/SMS Reference Manual contains full details of IO.EDLIN

## 15.7 HIS_SIZE

| | |
|---|---|
| Syntax | HIS_SIZE #channel, lines |
| Location | History Device |

A history which has been already been installed with HIS_SET has its size specified and is activated (or de-activated) with HIS_SIZE. The different usages of the command are distinguished by the lines parameter.

- Positive numbers for lines will activate a history for the given channel and tell it to store the next lines number of lines which are terminated by pressing <ENTER>. If a history had already been installed, then all stored lines are lost.
- Negative numbers will have exactly the same effect except that if a history was already active, the absolute value of the given lines number is added to the memory capacity of the existing history. Existing stored lines are retained in memory.
- Zero simply turns off a history and clears the tables which hold the entered lines.

History can store a maximum of 32767 lines which should be more than sufficient. Memory is allocated dynamically, in four kilobytes chunks, so there is a small danger of heap fragmentation.

**CROSS-REFERENCE**

See *HIS_SET*.

## 15.8 HIS_UNSET

| | |
|---|---|
| Syntax | HIS_UNSET #channel |
| Location | History Device |

This command removes a history from a channel, regardless of its state of activity and the stored lines. HIS_UNSET can only be used on channels where a history exists, otherwise an error will be reported.

**Example**

```
HIS_UNSET #0
```

**CROSS-REFERENCE**

*HIS_USE*, *HIS_SIZE*

## 15.9 HIS_USE

| | |
|---|---|
| Syntax | HIS_USE device$ |
| Location | History Device |

History's command line history is installed as a device driver to allow you to use it from languages other than SuperBASIC. The default device name is HIS and can be opened as an input pipe whenever a IO.EDLIN call is to be used.

The HIS_USE instruction allows you to rename this device name to any other three letter code, passed as a string. The use of the HIS device is beyond the scope of this book because it's not necessary for SuperBASIC where the HIS_SET, HIS_SIZE and HIS_UNSET commands are available to handle it. Please refer to the original documentation!

**SMS NOTE**

The History device built into SMSQ/E uses the device name HISTORY and can therefore be used alongside this version of the History Driver. You cannot rename SMSQ/E's version.

**CROSS-REFERENCE**

See *HIS_USE$* and *HIS_SET*.

## 15.10 HIS_USE$

| | |
|---|---|
| Syntax | HIS_USE$ |
| Location | History Device |

This function returns the three letter device name which has been set with HIS_USE.

**CROSS-REFERENCE**

See *HIS_USE*.

## 15.11 HOT

| | |
|---|---|
| Syntax | HOT key, executable_file |
| Location | TinyToolkit |

This command will load the given executable job into memory and start it running from memory each time that the specified key (together with <ALT>) is pressed, so there will not be any need to access the drive, but the code has to be stored twice: once as the code loaded by HOT, and then the job created from that code. Thus it is only practical to load small programs such as system utilities with this command.

**Examples**

```
HOT c,FLP1_COLOURS_exe
HOT s,FLP1_tk2flp
HOT "4",FLP1_QED
```

**NOTE 1**

Any ALTKEY definitions which use the same hotkeys will be ignored.

**NOTE 2**

Non-standard machine code cannot be used (Supercharged or Turbo compiled SuperBASIC for instance): the code has to be re-entrant, ie. when the job stops it should disappear. Jobs which relocate themselves, redefine the trap table, change their own code (ie. are not ROMable), or can only be started one at a time, tend to produce system crashes and other problems.

**WARNING**

Memory used by HOT-loaded programs cannot always be freed for use by SuperBASIC.

**CROSS-REFERENCE**

*CLEAR_HOT* clears a hotkey defined with *HOT* and (hopefully!) returns the occupied memory to QDOS. Use the Hotkey System if you have this available!!

## 15.12 HOT_CHP

| | |
|---|---|
| Syntax | HOT_CHP (key$,filename [;cmd$] [,JobName$] [,options]) |
| Location | HOTKEY II |

The main idea behind the Hotkey System II is that you can have access to any number of QL programs by pressing one simple hotkey in order to access each program, rather than having to use <CTRL><C> to cycle through all of the programs currently stored in the QL's memory.

The function HOT_CHP will load an executable file with the specified filename into the common heap and make it into an Executable Thing. Now, each time that you press <ALT> plus the specified key$, a new copy of the program will be started up in memory (although the same code is used, meaning that very little memory is used by each additional copy).

As from v2.03 of Hotkey System II, if you use an upper case key$, then you will need to press the upper case character, compare where you use a lower case key$, which will recognise both the upper and lower case character (if the upper case character has not been assigned to another hotkey).

HOT_CHP will support the current program default device if Toolkit II is loaded, otherwise it will use its own default device which can be configured by using the program CONFIG on the file HOT_REXT.

When the program is loaded using this command, HOT_CHP will look to see whether the start of the program contains a Job name, if not, then the program file name is used as the Job name (unless an alternative is stipulated, using the Jobname$ parameter).

As with EXEP, you can pass a command string to the program which will be passed to each copy of the program as and when they are started up. You can also supply a specific Job name for the program and pass various options to the Pointer Interface to tell it how to treat the program. As well as those options supported by EXEP, the following option is also supported:

- -I This tells the Hotkey System that the program code is 'impure' (ie. it modifies its own code). This means that code cannot be shared by every copy of the program - this therefore means that each time that the program is called, a copy of the original code is made from which the program runs. For this reason, you should consider using HOT_LOAD for such programs. The most common programs which fall within this category have been written under BCPL or compiled with Supercharge or Turbo.

If the program is successfully loaded into memory and set up as an executable Thing, HOT_CHP will return 0, otherwise one of the following error codes will be returned:

- -2 Specified filename is not executable
- -3 Not enough memory to load the file
- -7 The specified filename cannot be found
- -9 The specified hotkey has already been defined, or the file is in use.
- -12 The specified filename is not supported (bad filename).

**NOTE 1**

Any programs which are to be loaded into the Hotkey System II should be re-entrant so that the same code can be shared by any number of copies of the program, otherwise label them as Impure.

**NOTE 2**

Versions of the Hotkey System pre v2.21 do not allow you to pass a command string.

**WARNING**

You should not specify a Job name for impure programs as this may cause problems.

**CROSS-REFERENCE**

If you do not intend to remove the program in the future, use *HOT_RES* or *HOT_RES1* as these will ensure that the program starts up more quickly. *HOT_CHP1*, *HOT_LOAD*, *HOT_LOAD1* are similar. The hotkey will not be available until you enable the Hotkey System with HOT_GO.

## 15.13 HOT_CHP1

| | |
|---|---|
| Syntax | HOT_CHP1 (key$,filename [;cmd$] [,Jobname$] [,options] ) or<br>HOT_CHP1 (key$,filename [;cmd$] !Wakename$ [,options] ) |
| Location | HOTKEY II |

The first variant of this function is very similar to HOT_CHP except that it will only start up a new copy of the program when the specified hotkey is pressed if there is not already a copy of the program being executed. If a copy of the program is already being executed, then the hotkey will merely move that copy of the program to the top of the pile so that you can access it (it will PICK the program and execute a WAKE event, if supported by the program - a Wake event is normally used by a program to force it to update its tables etc).

The second variant of this command was introduced in v2.24 of the Hotkey System II and allows you to specify a name of a job (Wakename$) which is to be woken up if there is already one copy of the original program running in memory. Unfortunately this variant acts differently from the first in one main way:

- If the original program is already running, and Wakename$ points to another program which is not yet running, a second copy of the original program will be started up.

**Example**

The following line will allow you to set up the <ALT><R> key to do one of two things:

- If a job called QR-Config is running already, this will be Woken; otherwise;
- A copy of a program called flp1_Route_Obj will be started up (even if one is already running).

```
ERT HOT_CHP1 ('R','flp1_Route_obj';'flp1_\' ! 'QR-Config')
```

**NOTE 1**

On early versions of the Hotkey System II, HOT_CHP1 did not create an Executable Thing.

**NOTE 2**

Versions of the Hotkey System pre v2.21 do not allow you to pass a command string.

**CROSS-REFERENCE**

See *HOT_CHP*. *HOT_PICK* allows you to set up hotkeys to PICK a program, and *HOT_WAKE* allows you to set up hotkeys to WAKE a program. *HOT_THING* allows you to call an Executable Thing.

## 15.14 HOT_CMD

| | |
|---|---|
| Syntax | HOT_CMD (key$,command$ *[,command$]*) |
| Location | HOTKEY II |

This function allows you to set up a specify a key, which, when pressed with <ALT> will call up the SuperBasic task (Job 0), Picking it to the top of the pile, and then send each specified command to the command console (normally #0) followed by <ENTER> at the end of each string.

**Example**

```
ERT HOT_CMD ('d','INPUT "List Device: ";d$','DIR d$')
```

will set up a hotkey whereby whenever you press <ALT><d>, control will be returned to SuperBasic and the user asked to enter a device, after which, a directory of that device will be produced.

**NOTE**

Although HOT_CMD will quite happily allow you to redefine an existing hotkey created with HOT_CMD or HOT_KEY, if any other command has been used to set up the hotkey, error -9 (in use) will be reported.

**CROSS-REFERENCE**

See *HOT_KEY*. *HOT_GO* is required in order to make hotkey definitions operational. *FORCE_TYPE* is very similar.

## 15.15 HOT_DO

| | |
|---|---|
| Syntax | HOT_DO key$ or<br>HOT_DO Thingname$ |
| Location | HOTKEY II |

Once a hotkey is operational (see HOT_GO), you can call up the program or action set up on that hotkey by using the command HOT_DO, which enables a program to emulate the user pressing <ALT><key>.

The first variant expects you to supply the key which would normally be used together with <ALT> to call up the facility. You can however, also use the second variant to supply the name of an Executable Thing to be called up.

**Example**

Take the following hotkey:

```
100 ERT HOT_WAKE ('f',Files)
110 HOT_GO
```

The following would all have the same effect:

- Pressing <ALT><f>
- HOT_DO 'f'
- HOT_DO Files

**CROSS-REFERENCE**

See the other *HOT*... commands about setting up hotkeys.

## 15.16 HOT_GETSTUFF$

| | |
|---|---|
| Syntax | HOT_GETSTUFF$ [index] |
| Location | SMSQ/E |

This function returns the contents of the hotkey stuffer buffer. If given a parameter of 0, or no parameter, it will return the current contents of the stuffer buffer (like ALT-SPACE). A parameter of -1 gets the previous contents, like ALT-SHIFT-SPACE

**Examples**

```
result = HOT_GETSTUFF$     : REMark Return current contents of stuffer buffer
result = HOT_GETSTUFF$(0)  : REMark Return current contents of stuffer buffer
result = HOT_GETSTUFF$(-1) : REMark Return previous contents of stuffer buffer
```

**CROSS-REFERENCE**

See *GET_STUFF$*, *HOT_STUFF*.

## 15.17 HOT_GO

| | |
|---|---|
| Syntax | HOT_GO |
| Location | HOTKEY II |

The Hotkey System II is actually a Job (called HOTKEY) which sits in the background of the QL looking for the user to press the previously defined hotkeys. As many users should be aware, whenever a job is present in the QL's memory, you cannot access the resident procedure memory (which should be used to install SuperBasic extensions and device drivers for example - see RESPR).

For this reason, the Hotkey System II was designed so that the Hotkey Job would not actually be created until such time as the user was ready - ie. when all of the hotkeys had been defined and everything loaded into the resident procedure memory. Users who have used Toolkit II's ALTKEY system may have noticed that although they have defined various hotkeys (with HOT_KEY for example), they do not work (or as soon as the Hotkey System II has been loaded, the last line recall does not work). This is because the Hotkey Job has to be started. This is achieved simply by using the command:

```
HOT_GO
```

This will start the Hotkey Job which will support all of the currently defined hotkeys, including the Hotkey Stuffer Buffer keys (which can be re-defined by using the program CONFIG on the file HOT_REXT), and the last line recall. If you want to remove the Hotkey Job at any time, you can do so by using the command HOT_STOP, which has the same effect as RJOB 'Hotkey'. This will not destroy any of the Hotkey definitions and when you enter the command HOT_GO again, they will all be available once again.

**CROSS-REFERENCE**

*RESPR* allocates areas of the resident procedure memory.

## 15.18 HOT_KEY

| | |
|---|---|
| Syntax | HOT_KEY (key$,string$ [,string2$ [,string3$… ]]) |
| Location | HOTKEY II |

This function is very similar to the first variant of the command ALTKEY provided by Toolkit II, except that it operates by virtue of the Hotkey Job, rather than a polled task, which should make the hotkey a little more reliable than the Toolkit II version (although this does mean than a hotkey set up under the Hotkey System II cannot be accessed from within a program running in Supervisor mode).

As with ALTKEY, this function creates a key macro which will be typed into the current keyboard queue each time that you press <ALT> and the specified <key$> at the same time. Again, if more than one string appears in the definition, an <ENTER> (line feed) will be placed between each string. If you want a line feed at the end of the final string, add a null (empty) string to the definition.

**NOTE**

Although HOT_KEY will quite happily allow you to redefine an existing hotkey created with HOT_CMD or HOT_KEY, if any other command has been used to set up the hotkey (eg. ALTKEY), error -9 (in use) will be reported.

**CROSS-REFERENCE**

As with other Hotkey System II definitions, you will need to use *HOT_GO* before you can access this hotkey. See *ALTKEY* for more information.

## 15.19 HOT_LIST

| | |
|---|---|
| Syntax | HOT_LIST [#ch] or<br>HOT_LIST \filename |
| Location | HOTKEY II |

This command will produce a list in the given channel (default #1) of all of the currently set hotkeys recognised by the Hotkey System II. If the second variant of the command is used, this will create a file with the specified filename (default data device supported), offering the option to overwrite any existing file, and list the hotkeys in that file. Each hotkey will be listed in tabulated form, with the key (which has to be pressed together with <ALT>) followed by the operation or definition string. If you need to press <SHIFT> along with the key, the key will be pre-fixed with 's'.

**CROSS-REFERENCE**

*HOT_NAME$* returns the description or name for the hotkey. *HOT_TYPE* returns the type of hotkey operation.

## 15.20 HOT_LOAD

| | |
|---|---|
| Syntax | HOT_LOAD (key$,filename [;cmd$] [,JobName$] [,options] ) |
| Location | HOTKEY II |

This function is similar to HOT_CHP in the parameters which it expects. By contrast, however, HOT_LOAD does not store the program in memory, but, instead, each time that the specified hotkey is pressed, it will look for the specified filename and then load the program at that stage (this is therefore really designed for programs which are stored on Hard Disk, as it is improbable that you will keep the same disk in a drive all of the time).

**NOTE 1**

The I (Impure code) option is not needed with this function.

**NOTE 2**

HOT_LOAD does not create an Executable Thing.

**NOTE 3**

Versions of the Hotkey System pre v2.21 do not allow you to pass a command string.

**WARNING**

Versions of the Hotkey System II, earlier than v2.15 (or Level B-08 of the ST/QL Drivers) contained serious bugs in HOT_LOAD which could either remove the Hotkey Job or crash the computer.

**CROSS-REFERENCE**

See *HOT_LOAD1* and *HOT_CHP*.

## 15.21 HOT_LOAD1

| | |
|---|---|
| Syntax | HOT_LOAD1 (key$,filename [;cmd$] [,Jobname$] [,options] ) or<br>HOT_LOAD1 (key$,filename [;cmd$] !Wakename$ [,options] ) |
| Location | HOTKEY II |

This function bears the same relationship to HOT_LOAD as HOT_CHP1 does to HOT_CHP. See HOT_CHP1.

**NOTE**

Versions of the Hotkey System pre v2.21 do not allow you to pass a command string.

**CROSS-REFERENCE**

See *HOT_LOAD*.

## 15.22 HOT_NAME$

| | |
|---|---|
| Syntax | HOT_NAME$ (key$) |
| Location | HOTKEY II |

The function HOT_NAME$ returns the name of the Thing or the string associated with the specified hotkey. A null string is returned if the hotkey is not defined.

**Example**

```
ERT HOT_RES ('/',flp2_Qram): ERT HOT_KEY ('s','Yours Sincerely','')
HOT_GO
PRINT HOT_NAME$ ('/') , HOT_NAME$('s')
```

will show the following: Qram Yours Sincerely

**CROSS-REFERENCE**

*HOT_LIST* will list details about all currently defined hotkeys, *HOT_TYPE* allows you to verify the type of hotkey defined.

## 15.23 HOT_OFF

| | |
|---|---|
| Syntax | HOT_OFF (key$) or<br>HOT_OFF (Thingname$) |
| Location | HOTKEY II |

The HOT_OFF function allows you to turn off an individual hotkey by either specifying the hotkey itself, or the name of the Thing accessed by using the hotkey, if the second variant is used (if there are two hotkeys which access the same Thing, the first hotkey alphabetically will be turned off).

The second variant even allows you to pass the string or command used by HOT_KEY or HOT_CMD, although this is a somewhat dubious method of doing this!!

Even though the hotkey has been turned off, it will still appear in the hotkey list (see HOT_LIST), although pressing the hotkey will have no effect.

**NOTE**

If the hotkey or Thingname cannot be found, the function will return -7.

**Example**

```
HOT_OFF ('p')
```

will turn off the <ALT><p> hotkey, eg. if this is used by a program as a command.

```
HOT_SET ('p')
```

will turn it back on.

**CROSS-REFERENCE**

*HOT_SET* will turn the hotkey back on again. *HOT_REMV* will remove the hotkey definition for good.

## 15.24 HOT_PICK

| | |
|---|---|
| Syntax | HOT_PICK (key$, JobName$) |
| Location | HOTKEY II |

The function HOT_PICK is used to specify a hotkey to Pick a job of a specified name whenever that key is pressed together with <ALT>. In effect, whenever the hotkey is pressed, the specified program will be

brought to the top of the pile, allowing you to continue work on it. The Job Name given need only be the first word contained in the name shown when you use the JOBS command, therefore meaning that Job names can be as descriptive as you like! If the specified Job is not present in memory when you press the hotkey, a warning beep will be sounded.

**Example**

```
ERT HOT_PICK('p','Perfection')
```

will set up a hotkey which will allow you to jump straight into Perfection from any other program (provided that Perfection is in memory), just by pressing <ALT><p>.

**NOTE**

HOT_PICK up to v1.22 gave problems on the ST Emulators.

**CROSS-REFERENCE**

*EXEP*, *HOT_LOAD*, *HOT_CHP* and *HOT_RES* all allow you to alter the Job Name of a program as it is loaded. Compare *HOT_WAKE*.

## 15.25 HOT_REMV

| | |
|---|---|
| Syntax | HOT_REMV (key$) or<br>HOT_REMV (Thingname$) |
| Location | HOTKEY II |

The HOT_REMV function allows you to remove the hotkey definition associated with the specified key or, if you prefer, the hotkey associated with the specified Thing. If the hotkey refers to a program which has been loaded into the common heap (eg. with HOT_CHP), then this area of the common heap will also be released.

**NOTE**

Prior to v2.26 of the Hotkey System 2, if key$ was an upper case letter, then any hotkey associated with the lower case letter would also be removed.

**CROSS-REFERENCE**

See *HOT_OFF* for further details.

## 15.26 HOT_RES

| | |
|---|---|
| Syntax | HOT_RES (key$,filename [;cmd$] [,JobName$] [,options] ) |
| Location | HOTKEY II |

This function is the same as HOT_CHP except that the program is loaded into the resident procedure area, and cannot therefore be removed in the future. If the resident procedure area cannot be accessed (ie. if a task is already being executed), this function will access the common heap.

**CROSS-REFERENCE**

*HOT_CHP*.

## 15.27 HOT_RES1

| | |
|---|---|
| Syntax | HOT_RES1 (key$,filename [;cmd$] [,Jobname$] [,options] ) or<br>HOT_RES1 (key$,filename [;cmd$] !Wakename$ [,options] ) |
| Location | HOTKEY II |

HOT_RES1 is the same as HOT_CHP1 except that the program is loaded into the resident procedure area. If this cannot be accessed for any reason, the common heap will be used.

**CROSS-REFERENCE**

See *HOT_RES* and *HOT_CHP1*.

## 15.28 HOT_SET

| | |
|---|---|
| Syntax | HOT_SET (key$) or<br>HOT_SET (Thingname$) or<br>HOT_SET (newkey$,oldkey$) or<br>HOT_SET (newkey$,oldThingname$) |
| Location | HOTKEY II |

The first two variants of this function are the opposite to HOT_OFF in that they re-activate the specified hotkey. If the specified hotkey does not exist, the value -7 will be returned. By contrast, the second two variants allow you to re-define a hotkey by assigning a new key which is to replace the old key press. If the specified new hotkey already exists, -9 will be returned, and if the old hotkey cannot be found, the value -7 will be returned.

**Example**

```
10 ERT HOT_CHP ('p','flp1_Perfection')
20 HOT_GO
30 ERT HOT_SET ('L','p')
40 ERT HOT_WAKE ('p','Pick')
```

**CROSS-REFERENCE**

See *HOT_OFF* and *HOT_KEY*.

## 15.29 HOT_STOP

| | |
|---|---|
| Syntax | HOT_STOP |
| Location | HOTKEY II |

See *HOT_GO*!

## 15.30 HOT_STUFF

| | |
|---|---|
| Syntax | HOT_STUFF string$ |
| Location | HOTKEY II |

The Hotkey System II allows you to pass information to a program by using an area of memory known as the Hotkey Stuffer Buffer. The contents of this buffer can be placed into the current keyboard queue by pressing <ALT><SPACE> to read the last item to have been placed into the Stuffer Buffer, or <ALT><SHIFT><SPACE> to read the previous item to have been placed in the Stuffer Buffer.

The keys used to recall the Stuffer Buffers can be configured by using the program CONFIG on the file HOT_REXT.

Each item can by default be a maximum of 512 characters long (although this can be configured from between 128 and 16384 characters if you wish). Note that two of the characters are used to store the length of the Stuffer Buffer and must therefore be deducted from this setting.

The command HOT_STUFF allows you to place the specified string$ into the Stuffer Buffer so that it may be read by other programs. If the Stuffer Buffer was previously empty, both <ALT><SPACE> and <ALT><SHIFT> <SPACE> will return the same, however, if something was already in the Stuffer Buffer, this will be read by <ALT><SHIFT><SPACE>, and the new entry as <ALT><SPACE>.

**Example 1**

Place an address in the Stuffer Buffer:

```
HOT_STUFF '10 Hardacre Way' & CHR$(10) & 'Hardacre' &CHR$(10) & 'Newcastle'
```

**Example 2**

Presuming an empty Stuffer Buffer, after:

```
HOT_STUFF 'DIR flp1_'
```

the Stuffer Buffer would look like this:

```
<ALT><SHIFT><SPACE> --- DIR flp1_
<ALT><SPACE> --- DIR flp1_
```

If you then use:

```
HOT_STUFF 'DIR flp2_'
```

the Stuffer Buffer would look like this:

```
<ALT><SHIFT><SPACE> --- DIR flp1_
<ALT><SPACE> --- DIR flp2_
```

**NOTE**

HOT_STUFF “” caused various problems until SMS v2.73 - see GET_STUFF$. It could even crash compiled programs!!

**CROSS-REFERENCE**

*GET_STUFF$* allows a program to read the contents of the Stuffer Buffer. *HOT_LIST* will allow you to see the contents of the Stuffer Buffer. *HOT_GO* is required before <ALT><SPACE> or <ALT><SHIFT><SPACE> will work!

## 15.31 HOT_THING

| | |
|---|---|
| Syntax | HOT_THING (key$,Thingname$ [;cmd$] [,Jobname$] ) |
| Location | HOTKEY II |

The function HOT_THING allows you to define a hotkey which will start up a new copy of an Executable Thing whenever the hotkey is pressed (if the Thing is present at that stage). You can pass a command string to the Executable Thing and even change the name of the Job which will be created by passing Jobname$.

More and more utilities are being written for QDOS which are set-up as Executable Things (for example, most of the menus provided by QPAC2 are in fact Executable Things), which is a means of providing various resources which a program can make use of (if they are present).

Executable Things can be seen as an executable program stored in memory, several copies of which can be started up at any time, but the same piece of machine code will be used by all of the copies, meaning that very little memory is required for each additional copy.

**Example**

```
ERT HOT_CHP('p',flp1_Perfection,'Perfection WP')
ERT HOT_THING('P','Perfection WP')
```

Both <ALT><p> and <ALT><P> will now have the same effect.

**NOTE 1**

Thingname$ should contain the full name of the Thing, otherwise it will not be recognised.

**NOTE 2**

Versions of the Hotkey System prior to v2.21 do not allow you to pass a command string. You also need v2.24+ to pass a job name.

**CROSS-REFERENCE**

*HOT_CHP* and *HOT_RES* turn a file into an Executable Thing. *THING* allows you to test if a Thing is present.

## 15.32 HOT_THING1

| | |
|---|---|
| Syntax | HOT_THING1 (key$,Jobname$ [;cmd$] [,Jobname$]) or<br>HOT_THING1 (key$,Jobname$ [;cmd$] !Wakename$) |
| Location | SMSQ/E v2.50+ |

This command is exactly the same as HOT_WAKE.

**CROSS-REFERENCE**

See *HOT_WAKE*.

## 15.33 HOT_TYPE

| | |
|---|---|
| Syntax | HOT_TYPE (key$) |
| Location | HOTKEY II |

This function is useful to find out the type of hotkey associated with the specified keypress. The values returned by HOT_TYPE are as follows:

- -8 Hotkey for Last line recall
- -6 Hotkey for recall previous Stuffer Buffer
- -4 Hotkey for recall current Stuffer Buffer (HOT_STUFF)
- -2 Hotkey stuffs a defined string into the keyboard queue (HOT_KEY)
- 0 Hotkey PICKS SuperBasic and stuffs a command into #0 (HOT_CMD)
- 2 Hotkey DOES code
- 4/5 Hotkey executes a Thing (HOT_THING,HOT_RES,HOT_CHP)
- 6 Hotkey executes a File (HOT_LOAD)
- 8 Hotkey PICKS a Job (HOT_PICK)
- 10/11 Hotkey WAKES or executes a Thing (HOT_WAKE, HOT_RES1, HOT_CHP1)
- 12 Hotkey WAKES or executes a File (HOT_LOAD1)

**CROSS-REFERENCE**

*HOT_NAME$* returns the name of the Thing or the string being accessed.

## 15.34 HOT_WAKE

| | |
|---|---|
| Syntax | HOT_WAKE (key$,Jobname$ [;cmd$] [,Jobname$]) or<br>HOT_WAKE (key$,Jobname$ [;cmd$] !Wakename$) |
| Location | HOTKEY II |

Many programs which have been written to use the Pointer Environment will recognise what is known as a WAKE event - this defines something that the program should do once control is returned to the program, for example, updating its tables.

Whereas PICKing a job merely brings it to the top of the pile ready for use, when you WAKE a job, not only is it brought to the top of the pile, but also a WAKE event is executed (if supported). You should therefore WAKE any program which provides information on the current state of the computer or SuperBasic program for example.

The function HOT_WAKE allows you to set up a hotkey which will Wake the specified Jobname$ if a copy of the program is already being executed. However, if there is not already a copy of the specified Job being executed, the hotkey will then look for an Executable Thing with the same name as Jobname$

(which should therefore be specified in full), which, if found, will be executed by the hotkey, creating a new copy of the program.

As with the other hotkey commands, a command string can be passed to the program when it is executed (this will be ignored if the program is merely woken).

As with HOT_CHP1, HOT_RES1 and HOT_LOAD1, you can specify a Wakename$ which allows you to use the Hotkey to access two jobs, if at least the first Job (or Executable thing) exists then the Hotkey will do one of two things:

- If there is a current job called Wakename$, then this will be woken; otherwise;
- The first Job (or Executable Thing) will be Woken if it exists (or otherwise will be started up).

HOT_WAKE is ideally suited for programs where you would not want more than one copy to be executed at any one time (eg. a calendar program).

**Example**

Some users prefer to be able to have a choice between either Waking an existing copy of a program (or executing the first copy) and loading another copy of the program at a later stage. This can be achieved, for example, with:

```
ERT HOT_RES ('Q',flp1_QUILL,'QUILL')
ERT HOT_WAKE ('q','QUILL')
```

**NOTE**

Versions of the Hotkey System prior v2.21 do not allow you to pass a command string. You also need v2.24+ to pass a job name.

**CROSS-REFERENCE**

*HOT_PICK* allows you to define a hotkey to PICK an existing Job.

## 15.35 HPUT

| | |
|---|---|
| Syntax | HPUT [#ch] length [ , access [, type [, dataspace [, extra ]]]] |
| Location | SMSQ/E |

This command allows you to set the various parameters which are contained in the header of the file attached to the specified channel (default #3). The command will use the supplied parameters to set the required information. If the specified channel is not open to a file, then an Invalid Parameter error is reported. The information which can be set is as per HGET.

**NOTE**

You will not be able to compile this command with Turbo or SuperCharge due to the fact that it alters its own parameters.

**CROSS-REFERENCE**

*HGET* reads the file header information. See *FGETH$* for information about the file header. See also *HEADS* and *SetHEAD*.

CHAPTER

SIXTEEN

# KEYWORDS I

## 16.1 I2C_IO

| | |
|---|---|
| Syntax | I2C_IO (cmd$, res_len [,device [,param ]] ) |
| Location | Minerva extensions |

The Minerva MKII operating system comes complete with a battery backed clock and a small amount of on-board RAM (256 bytes) which can be used to store various details whilst a machine is switched off, using some of those details to dictate the state of the machine when it is first switched on (or re-set).

An on-board serial bus is also included which can be used to link add-on interfaces and can transfer data at speeds up to 100kbits per second.

Interfaces currently exist to allow the QL to drive motors (up to 4 amps), relay switches (up to 3 amps) and an Analogue to Digital converter.

The I2C_IO function allows you to access the battery backed clock, RAM and other interfaces provided by Minerva MKII, through what is known as the $I^2C$ bus. The results of the function will be returned by way of a string.

The cmd$ should contain a series of bytes which are sent to the $I^2C$ bus to be sent to the device pointed to by the other parameters. This is normally a byte which represents a command, followed by the parameters for that command.

For the battery backed clock and RAM supplied with Minerva MKII, there are only three commands which are required:

- CHR$(164) -Write param bytes to the specified device. The first byte to be written should in fact be the memory address to write to. Param can be altered by preceding the command character in the cmd$ by the number of bytes to write (eg. CMD$=CHR$(6)&CHR$(164)&CHR$(36)&'HELLO' will write the string 'HELLO' to memory address 36 in the RAM). If you only use this to write one byte, then this will merely set the memory address for access by further Write or Read commands.
- CHR$(188) -Read param bytes from the specified device. Again, you can precede this command character by the number of bytes to be read if you wish. The bytes which are read will be returned as the resultant string.
- CHR$(255) -This signifies the end of the command string.

The other parameters allowed by the function are:

- Res_len which signifies the expected length of the return string , which must not be too short!!

- Device signifies which device is to be accessed. The value 80 is used to access the battery backed RAM.
- Param depends upon the command being sent via cmd$.

The on-board RAM is allocated as follows:

| Byte | Meaning |
|---|---|
| 0-15 | Reserved for the clock and other things set by the configuration program. |
| 16-19 | QDOS version number (if this is different to the string returned by VER$, then the rest of the configuration data stored in the RAM will be ignored). |
| 20-23 | Warm reset value (as per CALL 390) to be used when the QL is re-booted. |
| 24-25 | Year*2+month DIV 10 (do not amend!) |
| 26-27 | Copy of locations 22 and 23 - this is used to reset the system if the the values in locations 20-23 do not make any real sense (do not amend) |
| 28-29 | Each bit in these two locations can be set to disable up to 16 plug-in ROMs linked to the QL (bit 7 of location 28 represents the ROM which appears at the top of the F1…F2 screen when the QL is reset, bit 6 represents the second ROM and so on). |
| 30 | NET station number |
| 31 | System Enhancements (equivalent to POKE !124!49,x) |
| 32 | SuperBasic Enhancements (equivalent to POKE \\212,x) |
| 33-34 | RESERVED |
| 35 | Length of boot string (0 to 128) |
| 36-163 | Boot string or user area |
| 164-251 | RESERVED |
| 252 | SER1 device (see below) |
| 253 | SER2 device (see below) |
| 254 | PAR device (see below) |
| 255 | RESERVED |

The bytes contained in locations 252 - 254 are intended for use by programs to find out if printers or modems are connected and what type they are. The values currently supported are:-

- 0: Nothing connected to this port
- 1-23: Printer type (as per SDUMP command)
- 253: Tandata Modem
- 254: Astracom 'Native' Modem
- 255: Astracom Hayes-Compatible Modem

**Example**

You can use this command to make the QL always start up by loading a specified program:

```
startup$ = CHR$(232) & 'LRUN win1_ROUTE_boot' & CHR$(10)
command$ = CHR$(164) & CHR$(35) & CHR$ (LEN (startup$)) & startup$ & CHR$␣
↪(255)
PRINT IC2_IO (command$, 0, 80, LEN (startup$) + 1)
```

**CROSS-REFERENCE**

Some expansion boards have their own in-built battery backed clock, which may need to be protected from programs which re-set the system clock for their own purposes using *SDATE*.

See *PROT_DATE*.

Because Minerva MKII's battery backed clock is read through the $I^2C$ bus, it cannot be affected by programs, unless you abuse the *I2C_IO* function!!

## 16.2 IDEC$

| | |
|---|---|
| Syntax | IDEC$ (value,length,ndp) |
| Location | Toolkit II, THOR XVI |

This function is exactly the same as CDEC$ except that it does not place commas between the characters to the left of the decimal point.

**CROSS-REFERENCE**

See *CDEC$*.

## 16.3 IF

| | |
|---|---|
| Syntax | IF condition ….. |
| Location | QL ROM |

This command is used to mark the start of yet another powerful SuperBASIC structure which allows a program to perform various functions dependent upon the status of a condition. The condition will always be interpreted as having either the value 1 (true) or 0 (false), using boolean logic if necessary. Such conditions may be simple, such as x=2 or complex, as in x=3 AND (y=1 OR y=2).

There are actually two forms of the SuperBASIC structure:

IF condition {THEN | :} statement *[:statement]* [:ELSE statement *[:statement]*]

or

IF condition [{THEN | :}] *[:statement]* … [ELSE] *[:statement]* … END IF

The first syntax represents in-line code, and the keyword THEN can either appear or be replaced by a colon (:). If the condition is true, the statements following THEN (or :) are executed, until the end of the line is reached. There is actually no need for a colon after THEN, for example the following are all the same:

```
IF x=1 : PRINT 'x is 1'
IF x=1 THEN PRINT 'x is 1'
IF x=1 THEN:PRINT 'x is 1'
```

If during processing of the statements following THEN, a corresponding ELSE keyword is found, the interpreter will search the line for the corresponding END IF, in which case control will jump to the statement following the END IF. If however, the line does not contain a corresponding END IF, as with all other types on in-line code, control will jump to the next program line.

On the other hand, if the condition is false, the interpreter will search the line for the corresponding ELSE, which, if found, will force control to jump to the first statement following ELSE. Control then just continues along the program line and to the next program line. Note that a colon must appear before the word ELSE, and although not strictly necessary after the word ELSE, it is advisable to place a colon after the ELSE keyword (see the Note below).

If ELSE does not appear, control is passed to the statement following the corresponding END IF, or if not present, the next program line.

The second syntax represents the much more flexible long-form of the IF..END IF statement. On the first line containing the IF condition, the keyword THEN may be replaced by a colon, or even omitted altogether. If the condition is true, control is passed to the next program line. If during interpretation, an ELSE statement is found, the interpreter searches for the corresponding END IF and passes control to the statement following this.

If the condition is false, the interpreter once again searches for a corresponding ELSE. If this is not present, then control is passed to the next statement after the corresponding END IF. If on the other hand, ELSE is present, control passes to the statement following ELSE (which may be on the same line as the ELSE keyword). There is no need to follow ELSE by a colon in this long form.

**Example 1**

A short program to move a cross around the screen, using the keys <N>orth, <S>outh, <E>ast and <W>est, press <ESC> to leave program:

```
100 WINDOW 448,200,32,16:PAPER 0:CLS
110 x=224:y=100:OVER 0:INK 7
120 CURSOR x,y:PRINT 'X':OVER -1
130 REPeat loop
140 dir$=INKEY$(-1)
150 old_x=x:old_y=y
160 IF dir$ INSTR 'nesw'
170 IF dir$=='n':IF y>0:y=y-1
180 IF dir$=='s' AND y<200-10:y=y+1
190 IF dir$=='e':IF x<448-6:x=x+1
200 IF dir$=='w' AND x>0:x=x-1
210 ELSE IF dir$=CHR$(27):EXIT loop:ELSE NEXT loop
220 END IF
230 CURSOR old_x,old_y:PRINT 'X'
240 CURSOR x,y:PRINT 'X'
250 END REPeat loop
260 OVER 0
```

Notice the use of both AND logic operators and second IF statements (these can be swapped around). Placing the check for the keys <N>, <E>, <S> and <W> within another IF statement increases the speed

of this routine, as the four statements in lines 170 to 200 do not need to be processed if another key is pressed.

**Example 2**

The whole program can be simplified a little by using boolean logic, by replacing lines 170 to 200 with the following:

```
170 IF dir$=='n':y=y-(y>0)
180 IF dir$=='s':y=y+(y<200-10)
190 IF dir$=='e':x=x+(x<448-6)
200 IF dir$=='w':x=x-(x>0)
```

This is about 2.5% quicker than the first example.

**Example 3**

On a Minerva ROM, the powerful and even quicker SELect ON statement could be used to make things even easier to understand, by replacing lines 160 to 220 with:

```
160 SELect ON dir$
170   ='n':y=y-(y>0)
180   ='s':y=y+(y<200-10)
190   ='e':x=x+(x<448-6)
200   ='w':x=x-(x>0)
210   =CHR$(27):EXIT loop
215   =REMAINDER :NEXT loop
220 END SELect
```

This is about 22.5% quicker than the first example. Don't worry that the 'X' disappears in a band across the screen as it is being moved - as soon as you take your finger off the button, you are okay! It is unknown why this phenomenon occurs..

**NOTE 1**

On ROM versions earlier than Minerva v1.92 (unless you have SMS), when using multiple in-line IF statements, you need to be very careful over the use of ELSE and the colon ':'. Although the following two lines have exactly the same effect:

```
IF x=0 : PRINT 'HELLO' : ELSE PRINT 'Bye'
IF x=0 : PRINT 'HELLO' : ELSE : PRINT 'Bye'
```

The following gives the interpreter problems:

```
10 x=0
20 PRINT x
30 IF x=0 : PRINT 'HELLO' : ELSE IF x=2 : PRINT 'GOODBYE' : END IF : x=x+1
40 x=x+2
50 PRINT x
```

This should make x=2 at line 40, but in fact x=3.

This is because the interpreter does not look for an END IF following the ELSE IF structure.

Compare this with what happens if line 30 is made to read:

```
30 IF x=0 : PRINT 'HELLO' : ELSE : IF x=2 : PRINT 'GOODBYE' : END IF : x=x+1
```

This is actually a bug in the interpreter rather than a feature, as adding more IF statements into line 30 would appear to rectify it! The answer therefore is to ensure that a colon appears after every ELSE (or compile the program).

**NOTE 2**

Another problem also exists with in-line IF…END IF statements - in the following program, line 100 is called twice when d=1 and only once if d<>1.

```
2 IF d=1:PRINT 'd is 1':ELSE :PRINT 'd is not 1':END IF :PRINT 'A simple test
↪':GO SUB 100
3 STOP
100 PRINT "Now this is peculiar!!":RETurn
```

The rule would appear to be that the first GOSUB/PROCedure call after the END IF contained in an in-line IF…ELSE…END IF structure is called twice PROVIDED that the fist condition of the IF..ELSE..END IF statement is true. Both Minerva v1.93+ and SMS cure this. Otherwise, set the IF..ELSE..END IF statement out over several lines.

**SMS NOTES**

The improved interpreter checks whether IF statements are valid constructs before RUNning or SAVEing a program and will report one of the following errors if there is a problem:

**Incomplete IF clause**

Normally appears where END IF has been omitted other than in the in-line format.

**Misplaced END IF**

There is no matching IF … clause

**Misplaced ELSE**

This error is normally reported if an ELSE statement has not been placed inside an IF…END IF construct.

**CROSS-REFERENCE**

*SELect ON* provides a much quicker (although less flexible) means of testing a variable. Other SuperBASIC structures are *WHEN condition*, *WHEN ERRor*, *DEFine PROCedure*, *DEFine FuNction*, *REPeat* and *FOR*.

## 16.4 IFORMAT

| | |
|---|---|
| Syntax | IFORMAT device_[name] |
| Location | ATARIDOS |

This command formats the specified device in IBM PS/2 disk format, giving it the specified name (if any). The only difference between this and AFORMAT is the way in which the boot sector is created. As with FORMAT, this will normally format a disk to the highest possible density - however, you can force it to format a disk as single-sided by making the last character of the filename an asterisk (*). However, some IBM compatible PC's are unable to read single-sided disks.

**NOTE**

Unfortunately, you cannot format 360K or 1.2M disks with this command.

**CROSS-REFERENCE**

See *FORMAT* and *AFORMAT*. Other commands added are *IQCONVERT*, *ADELETE* and *QCOPY*.

## 16.5 INARRAY%

| | |
|---|---|
| Syntax | INARRAY% (array[ { $ \| % } ] [ ,first ] ,tofind[ { $ \| % } ] ) |
| Location | INARRAY (DIY Toolkit - Vol Z) |

This function searches a given array for a specified value. The array can be of any type, a string (although this must only be two-dimensional), a floating point or integer (these latter two can be any number of dimensions, up to 15 !!). INARRAY% will then search the specified array for the given value (tofind) which must be a string, floating point or integer value, although it does not have to be the same type as the array itself provided that you could assign the value to the array, for example:

```
array%(10)='2020'
```

and:

```
PRINT INARRAY%(array%,'2020')
```

are okay, compare:

```
array%(10)='x'
```

and:

```
PRINT INARRAY%(array%,'x')
```

which both return an error.

The search is not case-sensitive and will also equate embedded numbers so that the strings '020' and '20.00' are seen as the same as '20'. Like the function SEARCH, the search is very fast.

The first parameter can be specified, which allows you to tell INARRAY% from which element onwards it should look (remember that the first element is indexed with 0).

The value returned by INARRAY% will be -7 if the value is not found in the specified array.

An error will be generated if tofind could not be coerced to the same type as the array.

An error will also be generated if the array contains more than 32768 entries.

If the search is successful, INARRAY% will return one value which represents the index of the entry. For strings and single dimensional arrays, this is easy - if the value returned is srch, then:

```
PRINT array(srch)
```

will show the value you searched for. However, where the array has more dimensions, you will need a little work to find out the entry referred to.

For example, take a three-dimensional array s%(10,20,30) - this contains 11*21*31 (7161) entries, with the first entry being index 0, this being s%(0,0,0) and the last entry being index 7160, this being s%(10,20,30).

If INARRAY% (s%,300) returned the value 32, this would be index number 32, equivalent to s%(0,1,1). This could be found out by using the formula for s%, where the value returned (index) points to s%(x,y,z), where:

```
z=index MOD (31*21) MOD 31
y=index MOD (31*21) DIV 31
x=index DIV (31*21) MOD 31
```

It is important to work from right to left along the list of array elements, alternating MOD and DIV for each entry.

**NOTE**

This function will not work in a program compiled with Turbo or SuperCharge.

**CROSS-REFERENCE**

Use *INSTR* to locate a sub-string in a string. See search which is similar.

## 16.6 INF

| | |
|---|---|
| Syntax | INF |
| Location | Math Package |

The function INF is a constant and holds the greatest number which can be used in SuperBASIC. It is a floating point number exactly equal to $2^{2047}$. If any value becomes greater than INF, an overflow will occur. The smallest possible value is -INF.

**CROSS-REFERENCE**

*MAXIMUM* and *MINIMUM* can also be used to return this value.

## 16.7 INK

| | |
|---|---|
| Syntax | INK [#ch,] colour or<br>INK [#ch,] colour1,colour2 [,pattern] |
| Location | QL ROM |

This command sets the ink colour used inside the given window ch (default #1). Since the advent of the Extended Colour Drivers under SMSQ/E v2.98+ the scope of colours accepted by this command has been much enhanced and depends upon the colour mode selected for the current program. As a result, the ink colour can be either a solid colour if the first syntax is used (in which case colour can be any integer in the range 0..16777215) or a composite colour made up of the three parameters supplied in the second syntax (colour1 and colour2 must both be in the range dictated by the current MODE).

Luckily, SMSQ/E allows you to include binary and hexadecimal numbers directly in programs (eg INK $f800) which may make the non-standard QL colours easier to use. The way in which colours are handled depends upon the operating system. On most systems, only the Standard QL Colour Drivers are supported. However, SMSQ/E v2.98+ can be used to access further Extended Colour Drivers by configuring SMSQ/E to start with them loaded, or using the start-up screen on QPC.

### 16.7.1 STANDARD COLOUR DRIVERS

**MODE 4 and MODE 8**

This applies to standard QL operating systems, or can be set under SMSQ/E v2.98+ with the command DISP_COLOUR 0,800,600 or by altering the configuration of the operating system. There are eight solid colours which have the following values (although only four of these colours are available in MODE 4):

| Value | MODE 8 colour | MODE 4 colour |
|---|---|---|
| 0 | black | black |
| 1 | blue | black (should be avoided) |
| 2 | red | red |
| 3 | magenta | red (should be avoided) |
| 4 | green | green |
| 5 | cyan | green (should be avoided) |
| 6 | yellow | white (should be avoided) |
| 7 | white | white |

The values in MODE 4 which are marked "should be avoided" can be used on standard QLs, but lead to compatibility problems when run under the Enhanced Colour Drivers (see below). Other integer values in the range 8 to 255 are allowed, but these are generally 'composite' colours and repeats of other values.

### 16.7.2 EXTENDED COLOUR DRIVERS

The following is a description of the various colour modes available under the Extended Colour Drivers provided by SMSQ/E v2.98+. These are available once SMSQ/E is configured to use the Extended Colour Drivers. DISP_COLOUR can be used to switch between the standard and extended colour drivers.

**QL Colour Mode**

This is selected with the command COLOUR_QL and is the default when a program is executed. For the purposes of INK, PAPER, STRIP etc commands, it provides the same colours as under the Standard QL Colour Mode (provided the standard colour=0 to colour=7 is used), except that MODE 4 programs can actually access all 8 colours not just the standard 4. However, the actual colours which represent each of the different values can be amended by changing the palette (see PALETTE_QL). This can be used, for example, to rectify programs which display the wrong colours because they presume INK 3 would always be the same as INK 2.

**8 Bit Colour Mode**

This is supported on the Aurora motherboard (not yet implemented) and QPC, QXL and the Q40/Q60. It is selected with COLOUR_PAL and allows colour to be in the range 0…255. This is the PAL value and is hardware independent - refer to Appendix 16 for a full list of the colours available.

The colours which represent each of the 256 values allowed can be amended by changing the palette (see PALETTE_8). For this mode, the INK parameter should be the PAL value listed in the table. If a stipple is required, the two composite PAL colours will need to be specified explicitly - see below.

**Native Colour Mode (8 or 16 bit colour)**

This should be supported on all implementations of SMSQ/E v2.98+ and is selected with COLOUR_NATIVE. The range supported by colour and the effects all depend upon the display hardware currently in use. As a result, under Aurora, it is similar to COLOUR_PAL in that it only supports 8 bit colours, but the colour is specified by the Native Colour Value instead of the PAL value. On the QPC, QXL and Q40/Q60, it supports 65536 colours as standard. The value required for INK, PAPER, STRIP

etc. depends upon the hardware in use - look at the tables in Appendix 16 for the appropriate hardware and then the Native Colour Value to use. It may be easier to use hexadecimal or binary to specify the colour, for example INK $F81F for magenta on QPC/QXL.

**24 Bit Colour Mode**

This is only supported on QPC (dependent on hardware). It is selected with COLOUR_24 and allows colour to be in the range 0..16777215. Due the values possible in 24 bit colour mode, it is essential that hexadecimal is used to describe colours. Colours are defined as a 3 byte value representing a value for red, green and blue respectively. For example, yellow would be INK $FFFF00.

**COMPOSITE COLOURS**

These are colours made up of either two or three values, for example:

```
INK 2,7
INK 1,7,2
INK $F800,$FDBF,1
```

Depending upon the combinations, they may not be displayed correctly on a television.

```
INK colour1,colour2
```

This creates a composite colour made up of the two given colours in a checkerboard pattern (stipple 3).

```
INK colour1,colour2,stipple
```

This creates a composite colour which is a mixture of the two given colours, and displayed in the given stipple pattern.

The values for stipple are:

| Value | Pattern |
|---|---|
| 0 | Dots |
| 1 | Horizontal stripes |
| 2 | Vertical stripes |
| 3 | Checkerboard |

If you wish to calculate the equivalent single parameter for Standard QL Colour Mode, you will need to set various bits of colour by referring to the following table (note that this cannot be used under the Extended Colour Drivers except under COLOUR_QL):

| Stipple | BITS 76 | BITS 543 | BITS 210 | Colour |
|---|---|---|---|---|
| Dots | 00 | 000 | 000 | Black |
| Vertical Lines | 01 | 001 | 001 | Blue |
| Horizontal lines | 10 | 010 | 010 | Red |
| Checkerboard | 11 | 011 | 011 | Magenta |
| | 11 | 100 | 100 | Green |
| | 11 | 101 | 101 | Cyan |
| | 11 | 110 | 110 | Yellow |
| | 11 | 111 | 111 | White |

**NOTE**

Turbo and Supercharge cannot compile the THOR's floating point colours as they expect all parameters to be integers. Use IO_TRAP instead, for example:

```
a=IO_TRAP(#ch,39,colour): REMark Sets the PAPER colour.
a=IO_TRAP(#ch,40,colour): REMark  Sets the STRIP colour.
a=IO_TRAP(#ch,41,colour): REMark  Sets the INK colour.
```

Unlike the PAPER command, if you use IO_TRAP here, you will also need to set the STRIP colour explicitly.

**THOR XVI NOTE**

The THOR XVI allows a total of 16 colours in MODE 12 in the range 0 to 7.5 (stipple will actually fall in the range 0…1023). If you add .5 to the normal colour, this switches on the THOR's intensity bit, meaning that for example, the resultant colour for INK 1.5 is somewhere between black and blue (ie. a very dark blue). You can also add .25 to each colour, which will result in a stipple mixture of colours (details unknown at present).

**CROSS-REFERENCE**

*PAPER* and *STRIP* also set colours within windows. *RMODE* can be used to read the current colour mode. *COLOUR_QL*, *COLOUR_PAL*, *COLOUR_NATIVE* and *COLOUR_24* will also affect the colours produced. *PALETTE_QL* and *PALETTE_8* can be used to change the palette of colours available. DISP_COLOUR can be used to switch from Extended Colour Drivers to Standard Colour Drivers. Also refer to *INVERSE*. Please also look at the QL Display appendix (Appendix 16 - *A16 The QL Display*).

## 16.8 INKEY$

| | |
|---|---|
| Syntax | INKEY$ ( [#chan,][timeout] ) |
| Location | QL ROM |

This function fetches a single character from the specified channel (default #0). If a timeout is specified, INKEY$ will wait for timeout frames (there are 50 frames per second in the UK, 60 frames per second in most other countries). If a character is read, the function will return straight away, otherwise, it will wait for the specified number of frames and then return. Timeout can be in the range -32768..32767. If a negative timeout is specified, INKEY$ will wait forever until a character is read from the specified channel. The default of timeout is 0 which means return immediately. A timeout is therefore not really necessary if INKEY$ is being used to access a channel opened to a file, as the data will either be there or not! If #chan is not an input channel (eg. scr_), error -15 (bad parameter) will be reported.

**NOTE 1**

Using timeouts allows programs to run at the same speed on all QL compatibles.

**NOTE 2**

It may be useful to clear the input buffer before trying to read a character from the keyboard (this prevents overrun on keys) - you can do this by using something along the lines of:

```
100 dummy=KEYROW(0)
110 key$=INKEY$(-1)
```

**CROSS-REFERENCE**

*INKEY$* is channel based, which means that it can be used safely in multi-tasking programs. *KEYROW* will read the keyboard even though the current Job is not the one executing the *KEYROW* command (although see the options available with *EXEP*). *INPUT* allows you to read a string of characters in one go. *PAUSE* halts program execution temporarily.

## 16.9 INPUT

| | |
|---|---|
| Syntax | INPUT [#chan,] $^{*}$[ [separator] [prompt$^{i}$$ separator] var$^{i}$]$^{*}$ or<br>INPUT $^{*}$[ [#chan,] [separator] [prompt$^{i}$$ separator] var$^{i}$]$^{*}$ (THOR XVI and Minerva v1.97+ only) |
| Location | QL ROM |

This command will read a string of bytes from the specified channel (default #1), which must end in CHR$(10) = <ENTER>. The fetched string is then placed in the specified variable (var), which may be of any type. Several sets of bytes may be read at the same time by specifying more than one variable, for example by:

```
INPUT a$,x,b$
```

Although each set of bytes must again be terminated by CHR$(10).

If the channel is write-only (eg. scr), error -15 (bad parameter) will be reported.

If the specified channel is a console channel (con), the cursor will be activated and the user will be able to type in a string of characters at the current text cursor position. The characters typed will appear in the current INK colour on the current STRIP colour, and will also be affected by the settings of CSIZE, UNDER, FLASH and OVER.

If a channel is specified, this must be followed by a comma. It may however also be followed by one or more separators. Each separator may be one of the following:

- ! - (Exclamation mark) If a character other than a space appears immediately to the left of the current text cursor position, print a space. If prompt$ is specified after this, if prompt$ is too long to fit on the line from the current text cursor position, it will be placed at the start of the next line.

  If nothing follows this separator, then the text cursor is not moved at the end of the command.

- , - (Comma) This forces the text cursor to be placed on the next column which is a multiple of 8. Note that anything which appears on screen underneath the columns which are stepped over will in fact be blanked out in the current STRIP colour. If the next column which is a multiple of 8 is at the end of the current line, then the comma will move the text cursor to the start of the next line, not overwriting any text on screen!

- \ - (Back slash) This forces the text cursor to be placed at the start of the next line. If nothing follows this separator this has no further effect - the text cursor is automatically placed at the start of the next line at the end of INPUT anyway (see below). This has no effect unless nothing follows this separator, in which case the text cursor is left alone at the end of the command.

- TO col - This moves the text cursor to the specified column (col). If however, the text cursor is already at or beyond the specified column, the text cursor is moved one space to the right (unless

you have a THOR XVI - see TO). This separator must however be followed by yet another separator (normally ; (semicolon) so as to avoid confusion). If the specified column is further than the far right side of the specified channel, then TO merely wraps around the channel, continuing to count from the start of the next line. Note that any text under the columns which are jumped by TO will be blanked out in the current STRIP colour.

At the end of the INPUT command, the text cursor is placed at the start of the next print line (unless an end separator of '!', '\' or ';' is used).

If prompt$ is specified, this will have no effect unless the specified channel (#chan) is a console channel. If this is the case, the specified string is written to the console channel, (as with PRINT), followed by the specified separator. The cursor on the specified channel is then activated at the current print position and input awaited as normal if required.

If you are wondering how to include a variable as part of prompt$, this is achieved by placing the variable in brackets, for example the following will prompt for 3 names to be entered:

```
100 DIM a$(3,10)
110 FOR i=1 TO 3
120   INPUT 'Enter name number' ! (i) ! a$(i) TO 40; '-- Thank you'
130 END FOR i
```

Unfortunately, you cannot include the variable which has been entered in that same INPUT statement as a part of prompt$. If you do so, the prompt$ will include the variable at the value it contained at the start of the INPUT statement. For example, the following will not work correctly, always saying x^2=1 no matter what value you enter:

```
x=1: INPUT #2 ; 'Enter Number to Square' ! x \ 'x^2=' ; (x^2)
```

This could be fixed by using the following:

```
x=1: INPUT #2 ; 'Enter Number to Square' ! x: PRINT 'x^2=' ; (x^2)
```

The keys available for editing the string of characters as you enter it (via a console window) are shown on the next page. Once the string has been entered, it is assigned to the specified variable and the interpreter then looks at the INPUT command to see if any further prompt$ need to be printed out, or whether any further variables need to be entered; and if so, will repeat the above steps.

**KEYS AVAILABLE FOR EDITING**

Once any prompt$ has been printed, whilst the user is inputting a string, the following keys are available to the user to edit the string being entered:

- <LEFT> Move cursor left one character (if possible)
- <RIGHT> Move cursor right one character (if possible)
- <ENTER> Accept string input
- <UP> Ditto
- <DOWN> Ditto
- <CTRL><LEFT> Delete character to left of cursor
- <CTRL><RIGHT> Delete character under cursor
- <CTRL><SPACE> Break current command - return control to #0

**Example 1**

```
INPUT #2, TO 10 ; x$ \ TO 10 ; y$ \ 'Name:' ! : INPUT #2, TO 10 ; a$
```

**Example 2**

A function which will return a numeric variable safely. This accepts leading and trailing spaces, and even spaces before the E part of a number. Unfortunately, there is no way to prevent overflow errors, where the number is outside the range 10E-616…10E616. However, it will accept for example: ‘ +1.32 E-20 ‘:

```
100 REMark Demonstration
110 AT 10,0:PRINT 'Enter number: ':no=INPUT_no(#1,10,13)
120 PRINT #0,lives
125 :
130 DEFine FuNction  INPUT_no (chan, posx, posy)
140   LOCal var$,ix,loop,er,E_pos,dota,c
150   er=0
160   REPeat loop
170     IF er<0:BEEP 1000,10:er=0
180     dota=0
190     AT#chan,posx,posy:PRINT#chan,FILL$(' ',20)
200     AT#chan,posx,posy:INPUT#chan,var$
210     IF var$="":er=-1:NEXT loop
220     FOR ix=1 TO LEN(var$)
230       IF var$(ix)<>' ':var$=var$(ix TO):EXIT ix
240     END FOR ix
250     FOR ix=LEN(var$) TO 1 STEP -1
260       IF var$(ix)<>' ':var$=var$(1 TO ix):EXIT ix
270     END FOR ix
280     IF var$(1) INSTR '.1234567890-+'=0:er=-1:NEXT loop
290     IF var$(1)='.':dota=1
300     E_pos='E' INSTR var$
310     IF E_pos+1>LEN(var$):er=-1:NEXT loop
320     IF E_pos=0:E_pos=LEN(var$)+1
330     FOR ix=2 TO E_pos-1
340       c=CODE(var$(ix)):IF c=46:dota=dota+1
350       IF c<>46 AND (c<48 OR c>57) OR dota>1:er=-1:NEXT loop
360     END FOR ix
370     IF E_pos>LEN(var$):RETurn var$
380     FOR ix=E_pos+1 TO LEN(var$)
390       IF var$(ix)<>' ':E_pos=ix-1:EXIT ix
400     END FOR ix
410     IF var$(E_pos+1) INSTR '1234567890-+'=0:er=-1:NEXT loop
420     IF var$(E_pos+1) INSTR '-+':IF E_pos+2>LEN(var$):er=-1: NEXT loop
430     FOR ix=E_pos+2 TO LEN(var$)
440       c=CODE(var$(ix)):IF c<48 OR c>57:er=-17:NEXT loop
450     END FOR ix
460     RETurn var$
470   END REPeat loop
480 END DEFine
```

**NOTE 1**

If you try to INPUT a string greater than 32766 characters, this may crash the system. It is therefore important that when INPUTting from a file which is longer than 32766 characters, you are certain that it contains a CHR$(10). If not, then use INKEY$.

**NOTE 2**

If no variable is specified, INPUT will have the same effect as PRINT. In particular, as from SMS v2.57, INPUT on its own will clear a pending newline, in the same way as PRINT on its own.

**NOTE 3**

Pre JS ROMs have a small input buffer, meaning that strings over 128 characters long lead to a 'Buffer Full' (-5) error. You can fix this for QLiberator with a compiler directive.

**NOTE 4**

INPUT a% cannot accept -32768 (except on Minerva v1.76+ and SMS).

**NOTE 5**

If you try to INPUT a value into a slice of an undimensioned string, the value will not be stored and BASIC may stop without a message. For example:

```
100 a$='Hello World'
200 INPUT a$(7 TO)
210 PRINT a$
```

The above program will not even attempt to allow you to INPUT the value. The cure on all ROMs is to dimension the string, or to INPUT a temporary variable:

```
100 a$='Hello World'
110 INPUT g$
120 a$(7 to)=g$
130 PRINT a$
```

**NOTE 6**

If the specified channel is not a console channel, prompt$ and any separators are completely ignored. If there is no data in the channel to be read, then the error 'End of File' (-10) is reported. Under SMS, the prompt$ is still printed out, but any attempt to read a variable results in 'Invalid parameter'.

**NOTE 7**

Except under Minerva v1.97+, if CURSOR was used to specify the position of the input line and the position specified could not be set with the AT comand, the display could become messy if the cursor was moved off the initial row and then returned to it.

**NOTE 8**

Before SMS v2.59 if the <BREAK> key was pressed during an INPUT command, the cursor could be left active.

**MINERVA NOTES**

Minerva provides the following additional keys for use in editing the string:

- <ALT><LEFT> move to start of current text
- <ALT><RIGHT> move to end of current text
- <TAB> move along to x*8th character from start of line (or end of data if nearer)

- <SHIFT><TAB> moves back in steps of 8 characters (or start of data if nearer)
- <CTRL><ALT><LEFT> delete to start of current (visible) line
- <CTRL><ALT><RIGHT> delete from current character to end of line
- <ESC> same as <CTRL><SPACE> (Break key)
- <SHIFT><ENTER> same as <ENTER>
- <SHIFT><SPACE> same as <SPACE>

Minerva v1.93+ alters keys further, both to make editing text easier and also to prevent some anomalies in earlier versions:

- <UP> where the input data consists of more than one line, the up key moves up a line, unless cursor on first line of data in which case ends input. Any lines which have scrolled up out of the window will be re-shown if you press <UP> to move onto those lines. On previous ROM versions, if a line had disappeared off the screen, you could not access it. The only downside to this, is that any prompt which appeared before the text being edited cannot be re-shown - the prompt is simply 'blanked out' in the current PAPER colour.
- <DOWN> where input data consists of more than one line, the down key moves down a line, unless cursor on last line of data in which case it ends the input. This will allow you to access data lines which have scrolled down out of the window.
- <SHIFT><RIGHT> moves you right to the start of the next word (or end of the data). The start of a word is taken to be where the character to left of the cursor is space and the character under the cursor is something other than space.
- <SHIFT><LEFT> moves you left to the start of the previous word (or start of the data).
- <CTRL> + any combination with <LEFT> or <RIGHT> will delete the characters moved over. Spaces to the right which are caused by deletions are cleared in current PAPER colour - all other versions clear in current STRIP colour. Minerva v1.96+ (as with THOR XVI) will also allow: DIM x(4):INPUT x

This will patiently ask you to input the five values of x(0) to x(4). Minerva v1.96+ (as per THOR XVI) also allows you to insert channel numbers part way through an INPUT statement, although - unlike the THOR XVI implementation - you still cannot use the variable entered as part of the output.

**SMS NOTES**

SMS provides the following additional keys for use in editing the string:

- <ALT><LEFT> move to start of current text
- <ALT><RIGHT> move to end of current text
- <TAB> move along to x*8th character from start of line
- <SHIFT><TAB> moves back in steps of 8 characters <ALT><LEFT>move to start of current text
- <ALT><RIGHT> move to end of current text
- <TAB> move along to x*8th character from start of line
- <SHIFT><TAB> moves back in steps of 8 characters
- <CTRL><DOWN> Deletes the whole of the input line

- <SHIFT><RIGHT> moves you right to the start of the next word (or end of the data). The start of a word is taken to be where the character to left of the cursor is space and the character under the cursor is something other than space.
- <SHIFT><LEFT> moves you left to the start of the previous word (or start of the data).
- <CTRL> + any combination with <LEFT> or <RIGHT> will delete the characters moved over.

The following keyings have also been altered:

- <DOWN> Has no effect!
- <UP> Has no effect!

**THOR XVI NOTES**

The THOR XVI (version 6.41) allows you to put channel numbers part way through a statement, for example:

```
INPUT 'Your name' ; #0 , name$ \ #1 ; ' is ' ; (name$)
```

instead of:

```
PRINT 'Your name ' ; : INPUT #0,name$ : PRINT 'is ';name$
```

The THOR XVI also allows you to INPUT arrays with one statement. For example:

```
DIM x(4): INPUT x
```

will wait around for five values to be entered. No other implementation (other than Minerva v1.96+) currently allows this.

**WARNING 1**

There is no facility to check the characters entered using INPUT and if someone tries to enter a non-numeric character when INPUTting a numeric variable an error will be caused. The second example provides a means of ensuring numeric input is entered safely. Also refer to CHECKF and CHECK%.

**WARNING 2**

You can crash SMS if you try to omit unwanted data by using the same variable more than once in the INPUT statement. For example, consider opening a channel to a file which contains a copy of a directory. The first two lines contained in the file are not needed, being the disk name and the number of sectors. You therefore may use a line similar to:

```
100 OPEN_IN #3,ram1_direc
110 INPUT #3,dummy$\dummy$
```

which would simply read these two lines. Unfortunately, on SMSQ/E (pre v2.88), this appears to corrupt the return stack and may cause problems when you try to use other variables. Minerva also exhibits some of the same traits, although it manages to avoid a system crash. Oddly, if you enter PRINT dummy$ following this command, will print either rubbish (on SMSQ/E pre v2.88) or the first line of the file (on Minerva), whereas it should in fact show the second line!! Even more curiously, if you RUN the program a second time, INPUT works correctly! Later versions of SMSQ/E act in the same way as Minerva. The original QL ROMs get this one correct.

**CROSS-REFERENCE**

The text cursor is positioned using commands such as *AT* and *CURSOR*. You may prefer to use *EDLINE$* which allows you to provide a default string for alteration, as well as specifying the maximum number of characters that can be typed in. *PRINT* has some similar characteristics. *HIS_SET* allows you to set a history for a console channel.

## 16.10 INPUT$

| | |
|---|---|
| Syntax | INPUT$ ([#ch,] length) |
| Location | BTool |

INPUT$ is identical to FREAD$ with the single difference that the function also stops reading if a line feed character CHR$(10) has been found. So, INPUT$ is dedicated to read line based text.

## 16.11 INSTR

| | |
|---|---|
| Syntax | str1$ INSTR str2$ |
| Location | QL ROM |

This operator searches str2$ for str1$ and if found, it will return the position of the first character of str1$ in str2$. The search is not normally case-sensitive. If str1$ cannot be found, the value 0 is returned.

**Examples**

```
s$='Hello World':PRINT 'world' INSTR s$
```

will print 7.

```
PRINT 'worlds' INSTR s$
```

will print 0.

**NOTE**

If str1$ is a nul string, eg. str1$="", INSTR will always return the value 1.

**SMS NOTE**

You can specify a case sensitive search using INSTR_CASE.

**CROSS-REFERENCE**

See *INSTR_CASE*. *INARRAY%* and search can be used to compare entries within arrays.

## 16.12 INSTR_CASE

| | |
|---|---|
| Syntax | INSTR_CASE flag |
| Location | SMSQ/E v2.58+ |

Normally the INSTR operator carries out a non-case sensitive search. This command allows you to specify that the search should be case sensitive (if flag=1) or revert to the old system (if flag=0).

**Example**

```
INSTR_CASE 1
PRINT 'Hello' INSTR 'HELLO'
```

will return 0

```
INSTR_CASE 0
PRINT 'Hello' INSTR 'HELLO'
```

will return 1

**NOTE 1**

This command does not seem to have any effect on the speed of the INSTR operator.

**NOTE 2**

The setting of INSTR_CASE is cleared (to non-case sensitive) following NEW, LOAD, MERGE, LRUN, RUN, MRUN.

**CROSS-REFERENCE**

See *INSTR*.

## 16.13 INT

| | |
|---|---|
| Syntax | INT(x) |
| Location | QL ROM |

This function returns the closest integer (as a floating point value) which is smaller than or equal to x. For positive parameters this means that the non-integer part of the number is cut off, so INT(12.75)=12 and INT(5)=5. Note that for negative numbers this is not true: INT(-12.75) = -13 because -13 < -12.75. INT should handle any number in the range -2^31<=x<2^31-2 and will otherwise return an ERR.OF, except under SMS where it can handle much larger numbers, in the range -2^255 <= x <= 2^2047.

Both SuperBASIC and Minerva MultiBasic will overflow on INT(2^31-1), the nominally highest 32 bit twos compliment number. The maximum for both Basics is INT(2^31-2), as indicated above.

The SuperBASIC range limitation can, under the pragmatic assumption that with not more than 9 significant digits the concept of a "fractional part" is already lost anyhow for such large numbers, be worked around by something along the lines of:

```
DEFine FuNction BigINT (x)
    IF ABS(x) > 2.1474E9 : RETurn x
    RETURN INT(x)
END DEFine BigINT
```

**Example**

A function Rond(x,d) to round a number x to d decimal places:

```
100 DEFine FuNction Rond(no,plac)
110   LOCal temp
```

(continues on next page)

(continued from previous page)

```
120   temp=INT(no*10^(plac+1)+.5)/10^(plac+1)
130   RETurn INT(temp*10^plac+.5)/10^plac
140 END DEFine
```

```
PRINT Rond (10.3226,2)
```

gives the result 10.32

**NOTE 1**

The INT function does not round to the nearest integer, use INT(x+.5) instead.

**NOTE 2**

On non-Minerva ROMs, unless you have SMS installed, INT with x>(2^31)-2 gives an overflow error.

**CROSS-REFERENCE**

*CEIL* is complementary to *INT*.

## 16.14 INTMAX

| | |
|---|---|
| Syntax | INTMAX |
| Location | Math Package |

The function INTMAX returns the constant 10737421823 = 2^30-1. Although SuperBASIC's integers can only handle a maximum of 16 bits (resulting in a range of $2^{16}$ different values: -32768 to 32767), the QL can internally, on a machine code level, happily handle larger integers. Many keywords listed in this book actually make use of this possibility and that explains their valid parameter range.

**NOTE**

An error in the Maths Package (up to v2.04) means that a line such as PRINT 2^30-INTMAX will report an out of memory error. This is fixed in later versions of the package.

**CROSS-REFERENCE**

*EPS*(x) = 1 if and only if *ABS*(x) = 2 * INTMAX.

## 16.15 INVERSE

| | |
|---|---|
| Syntax | INVERSE [ #channel ] |
| Location | ALIAS (DIY Toolkit - Vol A) |

This command provides the QL with a facility which is available on most other implementations of BASIC - inverse video. What this command actually does is swap over the values which have been set for the specified window channel (default #1) for the STRIP and INK colours, thereby making any further text which is PRINTed to that window stand out. This means that for example, if you set the INK to 7 (white) and the STRIP (or PAPER colour) to 2 (red), after INVERSE, text will be printed in red INK on a white STRIP.

**Example**

```
STRIP #2,7:INK #2,0:PRINT #2,'This text is normal'
INVERSE #2:PRINT #2,'But this is in inverse video!!'
```

**CROSS-REFERENCE**

See also *INK* and *STRIP*.

## 16.16 INVXY

| | |
|---|---|
| Syntax | INVXY x%, y% |
| Location | HCO |

This is a simple command which draws a crosshair on screen with its centre at (x%,y%). It is drawn with OVER -1 and uses the full screen.

**WARNING**

**Do not use this!**

## 16.17 IO_PEND%

| | |
|---|---|
| Syntax | IO_PEND% (#ch) |
| Location | BTool |

This function is the same as NOT PEND.

## 16.18 IO_PRIORITY

| | |
|---|---|
| Syntax | IO_PRIORITY level (level=1 to 1000) |
| Location | SMS |

This command is used to set the priority of the In / Out retry operations. This means that it affects the amount of time that the scheduler will spend retrying IN / OUT operations (such as INKEY$ or PRINT). Due to the QL's multitasking abilities, it is possible that a program can be running in the background whilst you are doing something else. If that program is trying to access a file or the Network port for example, then it may find that there is no information waiting to be read at the time and the scheduler will keep trying to access the file or Network until that information is received. By using this command to set a higher priority, the scheduler will allocate more time to doing this and less time to running another job. IO_PRIORITY 1is equivalent to the way in which the scheduler worked on the original QL ROM.

## 16.19 IO_TRAP

| | |
|---|---|
| Syntax | IO_TRAP ( [#ch], D0 [,D1 [,D2 [,A1 [,A2 [,D3 ]]]]] ) |
| Location | THOR XVI |

This function enables you to directly access QDOS's I/O TRAP (TRAP #3) machine code utilities. You need to supply the number of the TRAP call as the parameter D0, but you can also set certain of the other internal machine code registers used by the trap call by setting the other optional parameters. The TRAP call will affect the specified channel (default #1). Apart from parameter D3 (this defaults to -1), the purpose of the other parameters depends upon the TRAP call being used. Unless D3 is negative, 'Not Complete' errors will not be reported, thus allowing programs to continue even though their windows are buried under the THOR's windowing system and therefore unusable. The return parameter is the value returned in the machine code register D0 by the call.

**Example 1**

A short program to get the current window sizes and cursor position in the variables Window_Width, Window_Height, Window_posx and Window_posy respectively:

```
100 a = ALCHP (8)
110 dummy=IO_TRAP (11,0,0,a) : REMark IOW.CHRQ TRAP
120 Window_Width = PEEK_W (a): Window_Height = PEEK_W (a+2)
130 Window_posx = PEEK_W (a+4): Window_posy = PEEK_W (a+6)
140 RECHP a
```

**Example 2**

Switch on the cursor in #1 (call IOW.ECUR):

```
t = IO_TRAP ( HEX ('E') )
```

**Example 3**

Set cursor to column x in #3 (call IOW.SCOL):

```
t = IO_TRAP (#3,HEX ('11'), x )
```

**CROSS-REFERENCE**

*CLS*, *SCROLL* and *PAN* all allow you to access machine code trap calls on different ROMs. *INK* contains a good example of how to use *IO_TRAP*. *TTET3*,*MTRAP*, *QTRAP* and *BTRAP* are much better as they can be used on all ROM versions. The QDOS/SMS Reference Manual (Section 15) contains details of the I/O Access Traps.

## 16.20 IQCONVERT

| | |
|---|---|
| Syntax | IQCONVERT filename |
| Location | ATARIDOS |

This command takes a file which is stored on a QL Format disk and presumes that it was originally an IBM format file. It will then convert special characters in that file to QL compatible characters as well as converting any occurrence of a Carriage Return character (CR) followed by a Line Feed character (LF) to a single Line Feed character LF.

**CROSS-REFERENCE**

Compare *AQCONVERT* and *QICONVERT*. See also *IFORMAT* and *QCOPY*.

## 16.21 IS_BASIC

| | |
|---|---|
| Syntax | IS_BASIC |
| Location | MULTI |

The function IS_BASIC allows you to find out whether the SuperBASIC program which executes the command is running under the interpreter or has been compiled. This is done by returning the sum of the jobnumber and the jobtag: the sum is 0 for the interpreter and a positive number for a compiled job. So NOT IS_BASIC is 1 under the interpreter and 0 in a compiled program (or a MultiBASIC on Minerva or Multiple SBASIC under SMS).

**Example**

If a compiler is available, programs are normally compiled for: - faster loading - faster execution - possibly linking in Toolkits (QLiberator only) - easier multitasking - operating system independent code(QLiberator only) - shared code/hotkey execution (QLiberator only) IS_BASIC helps the programmer who uses the interpreter to develop programs which distinguish between features which are only available in compiled programs, for instance passing a command string:

```
100 IF NOT IS_BASIC THEN
110   CMD$="Test"
120 ELSE
130   INPUT CMD$
140 END IF
```

**NOTE**

IS_BASIC will fail to spot a MultiBASIC or SBASIC interpreter.

**CROSS-REFERENCE**

*PRIO* sets the priority of the current job. Under SMS or Minerva, you can use JOB_NAME$ to look at the name of the task which would normally be SBASIC or have its first two letters as MB respectively for a Multiple SBASIC or MultiBASIC interpreter, unless the name of the Interpreter has been altered. Refer to*JOB_NAME*.

## 16.22 IS_PEON

| | |
|---|---|
| Syntax | IS_PEON [{ #ch \| chan_ID \| job_name$ }] |
| Location | PEX |

This function takes the same parameter as PEON and returns 0 if PEX is not activated for the specified window (or Job) and returns a value not equal to 0 if PEX is active. If no parameter is specified, then this function just checks if PEX is active at all. If a pre-JS ROM is installed, then this function will return the value -19.

**CROSS-REFERENCE**

See *PEON* and *IS_PTRAP* for more details.

## 16.23 IS_PTRAP

| | |
|---|---|
| Syntax | IS_PTRAP (trapno [,status] ) |
| Location | PEX |

Not only does PEX allow you to enable background screen access for specific Jobs or windows, but you can also dictate how the various TRAP #3 machine code routines should be treated (which has a knock on effect on programs, since these routines are generally used to access the screen). For each TRAP #3 routine, you can specify the following status:

- 0 - if the window is buried, then halt the program when this routine is called (this is the normal case under the Pointer Environment)
- 1 - Enable background screen access for this routine (if PEX is active - see PEON).
- 2 - This only enables background screen access for this routine if both PEX is active and PXON has been used to enable SD.EXTOP routine calls.
- 3 - If the window is buried, then just ignore this call and allow the program to carry on. This could be used for example to allow a program which has a large amount of calculation to do to carry on in the background, printing a message to the screen only when its window is not buried to inform the user of its progress.

On JS and MG ROMs, only values of 0 and 3 are recognised - PEON activates all routines as having a status of 3 on these implementations. On all other implementations using the defaults provided with PEX, PEON activates all of the following routines as having a status of 1.

- $05 iob.sbyt
- $07 iob.smul
- $09 iow.xtop to $0B iow.chrq
- $0F iow.dcur to $36 iog.sgcr
- $6C iop.flim
- $72 iop.rpxl to $76 iop.spry

If you use PEX_SAVE, PEON will set the various routines as specified by you previously. Not all TRAP #3 machine code routines should be treated in this way - the following routines should be avoided if possible:

- $00 iob.test
- $01 iob.fbyt
- $04 iob.elin
- $0C iow.defb
- $0E iow.ecur

If you decide to use this function to fine-tune the operation of PEX, then you can save the various settings using the PEX_SAVE command. If status is not specified, then the value returned will be the status of that particular machine code routine. If a negative number is returned then you probably have a JS or MG ROM (or earlier). If you are writing a program which will is to run on all QLs, then you may wish to use IS_PTRAP to set all of the routines to 0 if the QL ROM version is JS, MG or earlier.

**NOTE**

You need a good book on the QL's operating system to be able to use this feature.

**CROSS-REFERENCE**

See *PEON* for more details. The QDOS/SMS Reference Manual Section 15 contains details of the various TRAP #3 calls.

CHAPTER

SEVENTEEN

# KEYWORDS J

## 17.1 JBASE

| | |
|---|---|
| Syntax | JBASE ( {jobnr \| jobname} ) |
| Location | TinyToolkit |

Each running job has a job header where information such as the job's priority is stored. Usually, the SuperBASIC programmer should read these settings by using JOBS, PJOB, OJOB etc, and change them with SPJOB, AJOB etc.

However, for some purposes it might be interesting to access a job header directly - hence this function.

The function JBASE takes either a job number or job name and returns the start address of where its job header is stored. JBASE breaks with error -2 (unknown job) if the parameter does not point to a job. The jobname need not appear in quotes (unless it is also the name of a SuperBASIC variable, procedure or function).

As from v1.11, the jobnr can be -1 which will return the base address of the current job.

The following parts of the job header are interesting from the SuperBASIC aspect, but please see system documentation for more details:

| Offset | Length | Meaning |
|---|---|---|
| 0..3 | 4 | Total length of job area |
| 4..7 | 4 | Job start address |
| 8..11 | 4 | Job ID of parent job (0 if none) |
| 12..15 | 4 | Address of job Released Flag (0 if Job released) |
| 16..17 | 2 | Job tag |
| 19 | 1 | Priority (Only on original QL ROMs and Minerva and Thor XVI) |
| 20..21 | 2 | Job status (0 active, >0 timeout, -1 suspended, -2 waiting) |
| 23 | 1 | Wait flag (bit 7 set if another job is waiting for this job) (not under SMS2) |
| 24..27 | 4 | Job ID of that waiting job |

**Example**

This program lists all running jobs by name, occupied memory and status:

```
100 CLEAR: CLS: id=0: UNDER 1
110 PRINT "Job";TO 20;"Size";TO 25;"Status"
```

(continues on next page)

(continued from previous page)

```
120 UNDER 0
130 REPeat job_list
140   id = NXJOB(id,0)
150   IF NOT id THEN EXIT job_list
160   name$=JOB$(id)
170   IF name$="" THEN name$="<anonymous>"
180   nr=id-65536*INT(id/65536)
190   base=JBASE(nr*(nr=nr))
200   length=1+INT(CVL(PEEK$(base,4))/1024)
210   status=CVI%(PEEK$(base+20,2))
220   SELect ON status
230     =0 TO 32767: timeout=INT(20*status)
240        status$="inactive for "&timeout&" ms"
250     =-1: status$="suspended"
260     =-2: status$="waiting"
270     =REMAINDER : status$="undefined"
280   END SELect
290   PRINT name$;TO 20;length;"k";TO 25;status$
300 END REPeat job_list
```

**NOTE 1**

JBASE returns an undefined value if the parameter used to be a job number but that job has already been removed. For example, create a job with CLOCK #1 and look up its number with JOBS:

```
Job Tag Owner Priority
0   0   0     32
4   8   0     s1 Clock
```

Enter the command:

```
PRINT JBASE(4)
```

or:

```
PRINT JBASE('clock')
```

and see the result.

Now kill the job with KJOB 4 and check with JOBS if it has really gone:

```
Job Tag Owner Priority
0   0   0     32
```

The job is dead but PRINT JBASE(4) still returns something - usually that number will be negative.

**NOTE 2**

JBASE sometimes behaves oddly due to rounding errors. Before v1.11, JBASE would report an 'invalid job' error if you used a variable to supply the job number (even if that variable pointed to an existing job). The example above shows how the problem can be easily circumvented: use JBASE (nr*(nr=nr)) instead of JBASE (nr) - this converts the variable into an expression.

**CROSS-REFERENCE**

*JOBS*, *SJOB*, *AJOB*, *NXJOB*. See *JobCBS* which corrects all of the problems associated with *JBASE*. Details of Job Headers can be found in the QDOS/SMS Reference Manual Section 18 p8.

## 17.2 JobCBS

| | |
|---|---|
| Syntax | JobCBS ( {jobnr \| jobname} ) |
| Location | BTool |

This function is identical to JBASE, but the problems mentioned in the notes above do not exist with JobCBS. Alas, it will not accept a jobnr of -1 in current versions.

## 17.3 JOBID

| | |
|---|---|
| Syntax | JOBID [({nr, tag} \| <name>)] |
| Location | SMSQ/E |

This function returns the 32-bit job id of the job with the given details as a decimal value. The optional parameters may be either a job number and job tag (as displayed by the JOBS command), or the job name. If no parameters are supplied, the Job ID number of the current job is returned.

**Examples**

```
result = JOBID            :REMark Returns the job ID of the current job
result = JOBID(job_name$) :REMark Returns the job ID of the job specified in␣
↪job_name$
result = JOBID$(nr, tag)  :REMark Returns the job ID of the job specified by␣
↪nr and tag
```

**CROSS-REFERENCE**

See *JOBS*, *JOB$*.

## 17.4 JOBS

| | |
|---|---|
| Syntax | JOBS [#channel] or<br>JOBS \file |
| Location | Toolkit II, THOR XVI |

This command lists all jobs currently loaded into the QL to the given channel (default #1). Five pieces of information are given: jobnumber (job), tag, owner, priority and jobname

**JobNumber / Tag**

The jobnumber and the jobtag are internally combined to form the job-ID (jobnumber+tag*2^16) in order to identify jobs.

**Owner**

The Owner of a job is not necessarily the job which started it nor must it be connected with it. If a job is removed, all jobs owned by it will disappear too.

**Priority**

The higher the Priority of a job, the more processor time is given to it and therefore the faster it runs.

An "s" in front of the priority means that the job is currently suspended, so the priority does not matter.

**Jobname**

The jobname is another method of identifying the job, being the name given to the job when it was programmed.

**NOTE 1**

If the second syntax does not work, you should update your Toolkit version.

**NOTE 2**

Minerva users will be dismayed to learn that current versions of this command do not display negative priorities.

**CROSS-REFERENCE**

Commands like *RJOB*, *SPJOB*, *REL_JOB*, *JOB_NAME* and *SJOB* change job settings; *JOB$*, *OJOB*, *NXJOB*, *PJOB* return the same information as appears on the *JOBS* list for single jobs. *LIST_TASKS* is similar.

## 17.5 JOB$

| | |
|---|---|
| Syntax | JOB$ (job_ID) or<br>JOB$ (jobnr,tag) or<br>JOB$ (jobname) |
| Location | Toolkit II |

This function returns the name of a specified job, or an empty string if it has no name or if the parameters do not specify an existing job. The third format is somewhat curious since you need to supply the very thing you are asking for (ie. the jobname)! A negative job_ID points to the calling job.

**CROSS-REFERENCE**

See *OJOB*, *PJOB*, *JOB_NAME* and *NXJOB* for more information on current jobs. Compilers normally include their own commands to set the correct job name.

## 17.6 JOB_NAME

| | |
|---|---|
| Syntax | JOB_NAME title$ |
| Location | SMS |

Although most Jobs have a name given to them by the programmer, there are three main exceptions to this rule:

- Minerva MultiBASICs have a job name beginning with the two letters 'SB' followed by a number.

- The main SuperBASIC interpreter has a job name equivalent to a null string ("") means that no name is shown under the JOBS command. QPAC 2 changes this in its menus to 'SuperBASIC' to identify this Job. You will also notice that when you put Job 0 to sleep (using <ALT><SHIFT><F1> ), under SMS the Button is given the name 'System'.
- SMS's multiple SBASIC interpreters are all given the name SBASIC and there is no way of distinguishing one copy from another.

This command allows you to set the name of a multiple SBASIC job under SMS to the specified title$. It has no effect on a compiled program, or on the main Interpreter (Job 0).

**Example**

It can be necessary to include code within a program which caters for different situations depending on whether the program is being run under an Interpreter or has been compiled. It is not sufficient to test the name of the job, nor the job number, as this can be different on various setups. Luckily, both Minerva and SMS allow you to discover whether a Job is an interpreter (other than Job 0).

- Minerva sets bit 6 in offset HEX('16') of the Job's Header;
- SMS places the four letters 'SBAS' at offset HEX('1E4') of the Job's header.

The following example uses these facts to decide what setup the program is running on:

```
100 vs$=VER$:prog_name$='TESTER'
110 IS_COMPILED=0:pass$=''
120 IF vs$<>'HBA' AND vs$<>'JSL1'
130 IF JOB$(-1)<>'':IS_COMPILED=1
140 ELSE
145   IF JOB$(-1)<>''
146     TJob=JBASE(-1)
147     IF vs$='HBA'
150       JOB_NAME prog_name$
160       IF PEEK$(TJob+HEX('1e4'),4)<>'SBAS': IS_COMPILED=1
165     ELSE
170       TByte=PEEK(TJob+HEX('16'))
180       IF NOT (TByte && 2^6): IS_COMPILED=1
182     END IF
185   END IF
190 END IF
200 IF IS_COMPILED
210   OPEN #1,con_448x200a32x16:PAPER 0:CLS
220   PRINT 'This program has been compiled'
230   pass$=cmd$
240 ELSE
250   IF JOB$(-1)<>''
260     OPEN #1,con_448x200a32x16
270   ELSE
280     WINDOW 448,200,32,16
290   END IF
300   PAPER 0:CLS:PRINT 'This program is running under an interpreter.'
310   IF JOB$(-1)=''
320     INPUT 'Enter Command String: ';pass$
330   ELSE
```

(continues on next page)

(continued from previous page)

```
340     pass$=cmd$
345     IF pass$='':INPUT 'Enter Command String: ';pass$
350   END IF
360 END IF
370 IF pass$='':pass$='UNDEFINED'
380 PRINT 'Command String was ';pass$
390 PAUSE
400 IF IS_COMPILED=0:IF JOB$(-1)=prog_name$:QUIT
410 IF IS_COMPILED=0:IF JOB$(-1)<>'':CLOSE #1
```

Unfortunately, we do not know of any way of testing whether a MultiBASIC or multiple SBASIC interpreter was started up using the EX command or not (if not, then the CMD$ will need to be entered).

**CROSS-REFERENCE**

See *SBASIC* and *MB*, about the multiple interpreters provided by Minerva and SMS. *JOBS* and *NXJOB* contain more information on current jobs. Compilers normally include their own methods of setting the correct job name. You may want to use *DEVTYPE* to test if a channel is open if a program is to run correctly under both Job 0 and a multiple SBASIC.

CHAPTER

EIGHTEEN

# KEYWORDS K

## 18.1 KBD_RESET

| | |
|---|---|
| Syntax | KBD_RESET |
| Location | ATARI_REXT |

This is a command which should never be needed. If you unplug the keyboard from the Atari whilst the machine is switched on (this can ruin your machine), when you plug it back in, you may find that the mouse buttons no longer work. This command re-initialises the keyboard driver so that the mouse buttons will work again!

## 18.2 KBD_TABLE

| | |
|---|---|
| Syntax | KBD_TABLE num or<br>KBD_TABLE kcode (SMS only) |
| Location | ST/QL (Level C-17 Drivers onwards), SMS (v2.31+) |

Various keyboards can be attached to a computer depending on which country the computer is being used in. It is therefore necessary to tell the operating system which keyboard layout is to be used so that it can recognise which keys are being pressed.

The command KBD_TABLE does just that. num will have one of six possible values depending on the keyboard layout (under SMSQ/E this equates to the international dialling code for that country) or you can use the second variant to pass up to four letters representing the Car Registration Letters for that country (the fourth letter is used where that country has more than one language):

| Num | Kcode | Language |
|---|---|---|
| 33 | F | French |
| 34 | E | Spanish |
| 44 | GB | English |
| 45 | DK | Danish |
| 46 | S | Swedish |
| 47 | N | Norwegian |
| 49 | D | German |

Num can also be used as a pointer to a user-defined keyboard table for countries which are not covered.

Details are beyond the scope of this book. A description of how to create new keyboard tables and languages appears in IQLR Volume 5 Issue 1 and 5.

**ST/QL NOTES**

The values for num must be one of the following values: 0 English 1 German 2 French 3 Norwegian 5 Danish 6 Spanish

**CROSS-REFERENCE**

*SET_LANGUAGE* is similar for the THOR XVI. *LANGUAGE* and *LANGUAGE$* allow you to enquire about a language. *LANG_USE* allows you to change the language used by the system. *TRA* allows you to change the output to a printer.

## 18.3 KBD_USE

| | |
|---|---|
| Syntax | KBD_USE [ser_port] |
| Location | XKBD |

This command ensures that any incoming data from the specified serial port (1 for ser1, 2 for ser2) is transformed into keystrokes. Thus other computers or 8 bit keyboards can be used as an additional external keyboard. Using the command without a parameter, or zero (eg. KBD_USE 0) closes the ser channel and stops this operation.

**NOTE**

This task is carried out in the background of any other programs, but does not actually create a job.

## 18.4 KBYTES_FREE

| | |
|---|---|
| Syntax | memory = KBYTES_FREE |
| Location | DJToolkit 1.16 |

The amount of memory considered by QDOS to be free is returned rounded down to the nearest kilo byte. See also *BYTES_FREE* if you need the answer in bytes. The value in KBYTES_FREE may not be equal to *BYTES_FREE*/1024 as the value returned by KBYTES_FREE has been rounded down.

**EXAMPLE**

```
kb_available = KBYTES_FREE
```

**CROSS-REFERENCE**

*BYTES_FREE*.

## 18.5 KEY

| | |
|---|---|
| Syntax | KEY keynr, string$ or<br>KEY keynr (KEYMAN only) or<br>KEY (KEYMAN only) |
| Location | FKEY, KEYMAN |

The KEY command allows you to install keyboard short-cuts: that means if a certain key (specified with keynr) is pressed the specified string$ will be typed into the current keyboard queue. The actual implementation of the command under the two Toolkits is different, in that the FKEY variant only allows keynr to be in the range 1..5 representing the five function keys (F1 to F5) (see table below).

The KEYMAN version allows keynr to be in the range 1…60 which represents the following key presses:

| Key Result | Key Result |
|---|---|
| 1 <F1> | 31 <CTRL><K> |
| 2 <F2> | 32 <CTRL><L> |
| 3 <F3> | 33 <CTRL><M> |
| 4<F4> | 34 <CTRL><N> |
| 5<F5> | 35 <CTRL><O> |
| 6<SHIFT><F1>, <F6> | 36 <CTRL><P> |
| 7<SHIFT><F2>, <F7> | 37 <CTRL><Q> |
| 8<SHIFT><F3>, <F8> | 38 <CTRL><R> |
| 9<SHIFT><F4>, <F9> | 39 <CTRL><S> |
| 10<SHIFT><F5>, <F10> | 40 <CTRL><T> |
| 11<CTRL><F1>, <F11> | 41 <CTRL><U> |
| 12<CTRL><F2>, <F12> | 42 <CTRL><V> |
| 13<CTRL><F3> | 43 <CTRL><W> |
| 14<CTRL><F4> | 44 <CTRL><X> |
| 15<CTRL><SHIFT><ESC> | 45 <CTRL><Y> |
| 16<CTRL><SHIFT><F1> | 46 <CTRL><Z> |
| 17<CTRL><SHIFT><F2> | 47 <CTRL><SHIFT><1> |
| 18<CTRL><SHIFT><F3> | 48 <CTRL><SHIFT><K> |
| 19<CTRL><SHIFT><F4> | 49 <CTRL><SHIFT><L> |
| 20<CTRL><SHIFT><F5> | 50 <CTRL><SHIFT><M> |
| 21<CTRL><A> | 51 <CTRL><SHIFT><N> |
| 22<CTRL><B> | 52 <CTRL><SHIFT><O> |
| 23 <CTRL><SHIFT><C> | 53 <CTRL><SHIFT><P> |
| 24 <CTRL><D> | 54 <CTRL><SHIFT><Q> |
| 25 <CTRL><E> | 55 <CTRL><SHIFT><R> |
| 26 <CTRL><F> | 56 <CTRL><SHIFT><S> |
| 27 <CTRL><G> | 57 <CTRL><SHIFT><T> |
| 28 <CTRL><H> | 58 <CTRL><SHIFT><W> |
| 29 <CTRL><SHIFT><I> | 59 <CTRL><SHIFT><X> |
| 30 <CTRL><SHIFT><J> | 60 <CTRL><SHIFT><Y> |

Once initiated, each time that the specified keying is pressed, the given string will be typed into the keyboard queue, (note there is a maximum of 80 characters). The KEYMAN variant allows the second syntax, KEY keynr, which allows you to check the current definition of keynr, which is then written to #0.

KEY without any parameters activates the key translation if necessary. KEY keynr, “” can be used to clear a definition with the KEYMAN variant.

**Example**

```
KEY 5,"STAT"&CHR$(10)
```

**NOTE 1**

In applications such as word-processors many of these key- presses are already used for other purposes and this may cause problems - if you do use KEY to set a function key and then load a program which uses those function keys, the program will first of all register that the function key has been pressed and will then receive a stream of other keypresses (ie. the defined string).

**NOTE 2**

Key is quite a common variable name and so there is a large danger of errors occurring if KEY has been loaded into the computer alongside a program which uses such a variable name - for example, the statement:

```
key = KEYROW(1)
```

will make the program stop with error -17, (Error in expression).

Another problem would exist if key was declared in a program as a BASIC procedure or function, for example:

```
DEFine PROCedure Key
```

in which case, this would overwrite the machine code definition and even the command NEW will not restore it, however all stored KEY definitions are kept active, and they can no longer be altered because the keyword KEY is no longer recognised by the system.

**NOTE 3**

Neither of these Toolkits should be linked into a Qliberated job, otherwise it is possible that the code used for KEY will exist twice in memory, which would crash the machine. Also, the Toolkits should not be loaded into a MultiBASIC or Multiple SBASIC unless you are certain that this Interpreter will never disappear.

**CROSS-REFERENCE**

*NOKEY* under KEYMAN is equivalent to *KEY* without parameters. See *ALTKEY* for a concept similar to this one here.

## 18.6 KEYROW

| | |
|---|---|
| Syntax | KEYROW (row) |
| Location | QL ROM |

This function is used to read the keyboard. It is not linked with a channel which means that it can be used by a job to read the keyboard whether or not that job has an active cursor. This is mainly only of use for programs which work in the background unless a certain key is pressed to bring them to life. If you try to use this command in a program to control the screen, then this will undoubtedly lead to screen corruption as the active program may itself be accessing the screen.

The main advantage which this function has when compared to INKEY$ is that it allows programs to recognise when the user is pressing several keys at once, such as the left and up cursor keys to move diagonally.

The function KEYROW is able to read several keys at once by using a keyboard matrix, where each row is numbered and each key is assigned to a certain row. If that key is pressed, then a bit is set in the integer value of that row to represent which key has been pressed.

For British QLs, the format of the matrix is:

```
Row 0| 7 4 F5 F3 F2 5 F1 F4
Row 1| DOWN SPACE \ RIGHT ESC UP LEFT ENTER
Row 2| " M £ B C . Z ]
Row 3| ; G = F S K CAPSLOCK [
Row 4| J D P A 1 H 3 L
Row 5| O Y - R TAB I W 9
Row 6| U T 0 E Q 6 2 8
Row 7| , N / V X ALT CTRL SHIFT
```

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

Please see the *A9 Character Set, Keyboard*.

The table reflects the physical keyboard layout and so may be different on other languages.

Unfortunately, only one row at a time can be read with KEYROW and so the keys are arranged into groups - for example, all of the cursor keys appear on one row, as do all of the function keys.

When the function is used, the supplied parameter row specifies which row of the matrix is to be looked at. The value returned will have bits set representing which keys (if any) in that row were being pressed.

When KEYROW is used, any characters in the type-ahead buffer are cleared, therefore, if you wish to avoid accidental input by the user of unwanted keys, you could use:

```
dummy = KEYROW(0): Quit$ = INKEY$(-1)
```

This will clear the type-ahead buffer and then wait for a new key to be pressed - this is essential where, for example, you are asking for confirmation that a program should be quit.

**Example 1**

If you were holding the <SHIFT> key down together with the left and up cursor keys down:

```
PRINT KEYROW(7)
```

would return the value 1 and:

```
PRINT KEYROW(1)
```

would return the value 6.

**Example 2**

The following program moves a cross about the screen, using the cursor keys - diagonal movement is allowed:

```
100 MODE 4
110 WINDOW 512,256,0,0:PAPER 0:CLS
120 WINDOW 448,200,32,16
130 INK 7:OVER -1
140 SCALE 150,-120,-75
```

(continues on next page)

(continued from previous page)

```
150 x=0:y=0
160 LINE x-10,y TO x+10,y,x,y-10 TO x,y+10
170 REPeat loop
180   ax=KEYROW(1)
190   IF NOT ax:NEXT loop
200   LINE x-10,y TO x+10,y,x,y-10 TO x,y+10
210   IF ax&&2:x=x-(x>-120)
220   IF ax&&16:x=x+(x<128)
230   IF ax&&4:y=y+(y<75)
240   IF ax&&128:y=y-(y>-75)
250   IF ax&&8:PRINT 'Program Escaped':OVER 0:STOP
260   LINE x-10,y TO x+10,y,x,y-10 TO x,y+10
270 END REPeat loop
```

**NOTE 1**

Except under SMS, if you are holding three keys down which form three corners of a rectangle on the keyboard matrix, the KEYROW function will return the same value as if the key which appears in the fourth corner of the rectangle was also depressed. The QL Manual suggests this does not happen where one of the keys is <CTRL>, <ALT> or <SHIFT>, but this still happens on the QL, even with Hermes.

**NOTE 2**

Some replacement full-sized keyboard interfaces will not recognise where two letter keys, two function keys or two numerical keys are held down at the same time.

**NOTE 3**

It is just possible that if KEYROW is being executed whilst a task is being loaded or unloaded, the system will crash (the command does not take place in supervisor mode). This has been fixed on THORs v4.16 (or later) and Minerva.

**NOTE 4**

KEYROW had various problems under SMS before v2.58.

**NOTE 5**

This command is only partly implemented on THORs v4.16 (and later), and does not work at all on earlier versions: the only multiple keystrokes recognised are: <SHIFT>, <CTRL>, <ALT> and one other key! The corners of the numeric pad act as diagonal cursor keys and the <CAPSLOCK> result is obtained by pressing <5> on the numeric pad. INKEY$ should be used instead.

**NOTE 6**

It is highly recommended that KEYROW is *not used* at all.

It reads the physical keyboard directly, so will conflict with other jobs running in a multitasking environment.

Another problem is the differences between keyboards, for example, on a German QL the <Z> and <Y> keys are swapped over compared to a British QL - INKEY$ is much better!

**ST/QL NOTES**

On the ST/QL Emulator, the KEYROW table is much extended, with the following rows being added, both to take account of the additional keys available on an ST keyboard and also to take account of the numeric keypad (hence the repetition of various keys!):

```
Row 8 | F10 F9 F8 F7 F6 BACKSPACE ~
Row 9 | + - CLR/HOME
Row 10| DELETE INSERT
Row 11| <
Row 12| 8 7 * / ) ( HELP UNDO
Row 13| 0 3 2 1 6 5 4 9
Row 14| ENTER .
```

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

The additional keys have the following meanings:

- BACKSPACE Delete one character to left (CTRL left).
- SHIFT BACKSPACE Deletes a word to the left.
- DELETE Delete character under cursor (CTRL right).
- SHIFT DELETE Deletes word under cursor.
- UNDO Undo current operation (CTRL ALT up).
- SHIFT UNDO Mega undo!
- CLR/HOME Freezes screen (CTRL F5)
- INSERT Hard space (SHIFT SPACE)
- CTRL SHIFT ALT UNDO Hard reset.

The keys on the numeric keypad have been redefined as follows:

| Keying | Result |
|---|---|
| SHIFT ( | [ |
| SHIFT ) | ] |
| SHIFT / | |
| SHIFT ENTER | = |
| CTRL ( | { |
| CTRL ) | } |
| CTRL \ | \| |

**CROSS-REFERENCE**

*INKEY$* and *INPUT* allow user input from the keyboard whilst retaining compatibility across the various QDOS implementations. These commands are also job specific, and hence much better for use in multitasking programs. Some of the HOT_XXX commands and *EXEP* allow you to pass a U option to the program to force the computer to freeze all other programs whilst this one is running - this can be used to ensure that a program which uses *KEYROW* does not carry on reading the keyboard whilst you are using another program.

## 18.7 KEYW

| | |
|---|---|
| Syntax | KEYW (keyword$) |
| Location | Fn |

This is a function which can be used to find whether or not a given keyword is linked into the interpreter. If the keyword is known to the interpreter, KEYW returns 0, otherwise -7.

**Example**

The following function returns 1 on Gold Cards and 0 on other systems (or some early Gold Cards if they do not support the SLUG command). It will unfortunately always return 1 under SMS which has the SLUG command built-in.

```
100 DEFine FuNction Gold_Card
110   RETurn NOT KEYW("FLP_DENSITY") + KEYW("SLUG")
120 END DEFine Gold_Card
```

A better way would be to use the MACHINE function.

**NOTE**

This will always access Job 0, therefore it cannot tell you whether or not a keyword is linked into a MultiBASIC interpreter or a multiple SBASIC interpreter.

**CROSS-REFERENCE**

*ELIS*, *DEFINED* and *KEY_ADD* are similar. Compare *LOOKUP%*.

## 18.8 KEY_ADD

| | |
|---|---|
| Syntax | KEY_ADD (keyword$) |
| Location | BeuleTools |

This function returns the start address of where the machine code routine for the given keyword is stored. If the keyword is unrecognised by the system, 0 will be returned. This command will work from Multi-BASICs and Multiple SBASICs.

**Example 1**

The file server functions of Toolkit II will only work if the code is in ROM, on an EPROM cartridge, under SMS or on a Trump Card for example.

Nevertheless, the FSERVE command is also found in the configurable software version. You can easily check what version of Toolkit II is present by looking at the base address of TK2_EXT and then adapt your program to take account of the system configuration.

```
100 TK2_location=KEY_ADD("TK2_EXT")
105 FILE_SERVER=0
110 SELect ON TK2_location
120   = 48*1024 TO 64*1024: FILE_SERVER=1
130   = REMAINDER: IF VER$='HBA': FILE_SERVER=1
```

(continues on next page)

(continued from previous page)

```
140 END SELect
150 :
160 IF FILE_SERVER THEN
170   FSERVE
180 ELSE PRINT "Sorry, no ROM based file server."
190 END IF
```

**Example 2**

KEY_ADD does not produce an error (unlike ELIS) when a keyword is not found, and is therefore ideal to check if a certain Toolkit, command or function is present. This can be used within programs to adapt to available facilities.

In this example the internal date and time is saved as a hexadecimal number (this is more precise) if the necessary functions are present, otherwise it saves it as a floating point number.

The load routine then checks if the DATE was stored as a hexadecimal or decimal number, taking whatever action is necessary.

```
100 DateFile$="flp1_LastDATE_dat"
110 :
120 DEFine PROCedure WRITE_DATE
130   IF KEY_ADD("OPEN_OVER") THEN
140   OPEN_OVER#3,DateFile$
150   ELSE OPEN#3,DateFile$
160   END IF
170   IF KEY_ADD("HEX$") AND KEY_ADD("HEX") THEN
180     PRINT#3,"$" & HEX$(DATE,32)
190   ELSE PRINT#3,DATE
200   END IF
210   CLOSE#3
220 END DEFine WRITE_DATE
230 :
240 DEFine PROCedure READ_DATE
250   LOCal d$
260   OPEN_IN#3,DateFile$
270   INPUT#4,d$: CLOSE#3
280   IF d$(1)="$" THEN
290   IF NOT KEY_ADD("HEX") THEN RETurn
300   d$=HEX(d$(2 TO))
310   END IF
320   ADATE d$-DATE
330 END DEFine READ_DATE
```

**CROSS-REFERENCE**

*ELIS* differs from *KEY_ADD* in that it returns zero if the keyword is not found, rather than an error message. See also *KEYW* and *LOOKUP%*. *FIND* and *FLIS* are also useful.

## 18.9 KEY_RMV

| | |
|---|---|
| Syntax | KEY_RMV keyword$ |
| Location | Beuletools |

This is the same as *ZAP*!

## 18.10 KILL

| | |
|---|---|
| Syntax | KILL |
| Location | BeuleTools, KILL |

This command will cause all current jobs, except the main SuperBASIC interpreter (Job 0), to be stopped and removed from memory.

**Example**

The presence of jobs prevents the commands RESPR / LRESPR from grabbing memory in the Resident Procedure Area (although some implementations overcome this restriction, they do so by by allocating space in the Common Heap) and because some machine code programs cannot or should not be loaded into the common heap, the command KILL can be of assistance.

**WARNING**

Be sure that no important data gets lost!

**CROSS-REFERENCE**

*RJOB* removes single jobs, *KJOBS* works in the same way as *KILL*.

## 18.11 KILLN

| | |
|---|---|
| Syntax | KILLN |
| Location | Beuletools |

This command removes all current jobs and re-activates the following jobs if the respective facilities are available: HOTKEY (Qjump's Hotkey System) BLANK (Blanks screen if no key pressed) Qmons Nix-Job (see NIX) Server (see FSERVE)

**WARNING**

Again, be careful!

**CROSS-REFERENCE**

*KILL* removes all jobs as does *KJOBS*. *KILL_A* removes all jobs and clears the whole system for SuperBASIC.

## 18.12 KILLSOUND

| | |
|---|---|
| Syntax | KILLSOUND |
| Location | SMSQ/E for Q68 |

Kills (stops) the sound from the SOUNDFILE commands.

**Note 1**

This also removes the first job called "SOUNDFILE JOB" that it can find. Normally this is the only sound playing job as it is inadviable to have multiple sound sources playing all at once.

**CROSS-REFERENCE**

*SOUNDFILE*, *SOUNDFILE2*, *SOUNDFILE3*

## 18.13 KILL_A

| | |
|---|---|
| Syntax | KILL_A |
| Location | Beuletools |

This command forces everything which can be accessed from SuperBASIC to be cleared, killed or removed: Jobs, file definition blocks (except if Qjump's Hotkey System is present), variables, the DATA pointer, the common heap and all channels are closed. On an Atari QL-Emulator and under SMS, the buffer for the serial and parallel port is also cleared.

**WARNING**

Be very careful! All data will be lost. Avoid this command if you can: it is more like a safe emergency break.

**CROSS-REFERENCE**

*KILL*, *KILLN*, *KJOBS*, *DEL_DEFB*, *CLCHP*.

## 18.14 KJOB

| | |
|---|---|
| Syntax | KJOB jobname (TinyToolkit only) or<br>KJOB jobnr or<br>KJOB jobnr,tag (BTool only) |
| Location | TinyToolkit, BTool |

This command kills the given job (causing it, together with all of its owned jobs, channels and memory to be removed or freed). Jobname in the first variant can be passed as a string or as just the name of the Job without quotes (so long as that name is not defined as a variable or SuperBASIC PROCedure or FuNction). jobnr is the Job number as listed by the JOBS command. If this is -1, this will kill the current Job.

The last variant is surplus at present - it was intended that it would pass the second parameter back to the owner of the job, but due to an error in the code, this second parameter is seen as the job tag.

**Examples**

```
KJOB 'Perfection v2.04'
KJOB 1
```

**NOTE**

Before v1.11 of TinyToolkit, you could not pass jobnr as -1 nor could you use a variable to pass the jobnr (see JBASE).

**CROSS-REFERENCE**

*RJOB* has a slightly different syntax. *KILL* and *KJOBS* remove all jobs. *SPJOB*, *SJOB*, *REL_JOB*, *AJOB* are other commands which handle jobs. The function *JOB$* will return the name of the given job.

## 18.15 KJOBS

| | |
|---|---|
| Syntax | KJOBS |
| Location | TinyToolkit, BTool |

This is the same as *KILL*.

CHAPTER

NINETEEN

# KEYWORDS L

## 19.1 LANG_USE

| | |
|---|---|
| Syntax | LANG_USE num or<br>LANG_USE kcode |
| Location | SMS |

This command sets the language to be used by SMS for its message tables (this includes interpreter messages and error messages). The value of num and kcode can be the same as for the SMS implementation of *KBD_TABLE*. A description of the message tables and how to link in new message modules is contained in IQLR Volume 5 Issue 1 and Issue 5.

**NOTE**

If you set a different language to the version of the Psion programs which you are using, then you may find that the DATE function in Archive and Abacus fails.

**CROSS-REFERENCE**

*LANGUAGE* and *LANGUAGE$* allow you to find out about the current language. *TRA* allows you to set the various message tables also. See *KBD_TABLE* and *SET_LANGUAGE*.

## 19.2 LANGUAGE

| | |
|---|---|
| Syntax | LANGUAGE [(code)] |
| Location | SMS |

This function returns a number representing the international dialling code for the current language implementation (if code is not specified). Otherwise it will return the dialling code of the language which would be used if the language represented by code was installed using LANG_USE (in which case code can be either the international dialling code or the car registration code).

**CROSS-REFERENCE**

*LANGUAGE$* returns the car registration code. *LANG_USE* allows you to set the language for the messages.

## 19.3 LANGUAGE$

| | |
|---|---|
| Syntax | LANGUAGE$ or<br>LANGUAGE$ (code)(SMS only) |
| Location | THOR range of computers, SMS |

This function returns a string representing the current language layout of the keyboard which is linked into the QL. Unfortunately, the string returned is different on THORs and SMS's. For a list of the strings returned on THOR computers, see *SET_LANGUAGE*. Under SMS the string returned is the international car registration code for the language currently loaded (if code is not specified). The second variant returns the car registration code of the language which would be used if the language represented by code was installed using LANG_USE (in which case code can be either the international dialling code or the car registration code).

**CROSS-REFERENCE**

*SET_LANGUAGE* allows you to alter the current keyboard. See also *LANGUAGE* and *LANG_USE*.

## 19.4 LAR

| | |
|---|---|
| Syntax | LAR file, array |
| Location | ARRAY |

LAR loads a file which must have been stored with SAR or SARO into a dimensioned array. The array must have been initialised with DIM to the same dimensions of the stored array, and of course the type (float, integer, string) must be the same. The default data device is supported.

**CROSS-REFERENCE**

See *DATAD$* about the default data device. *SAR* and *SARO* are complementary commands.

## 19.5 LBYTES

| | |
|---|---|
| Syntax | LBYTES device_filename, start or<br>LBYTES [device_]filename, start(Toolkit II only) or<br>LBYTES #channel, start(SMS only) |
| Location | QL ROM, Toolkit II |

This command loads a chunk of machine code (or data) stored on the given device under the specified filename and will report the error 'Not Found' (-7) if either the device or filename does not exist. If Toolkit II is present, this command supports the default data device (see DATAD$). If found, the chunk of machine code is loaded into the QL, starting at the specified start address. The code is loaded in one huge block, which means that loading is very quick. However, there is also no check on the type of file being loaded and therefore you should make sure that you know what you are doing. Under SMS the third variant allows you to load the data from the specified channel which must be open to a file. This allows for more efficient programs, so that you can perform various tests on the file beforehand (such as test its length and file type), whilst only opening a channel to the file once.

**Examples**

One of the main uses of this command is to load in machine code Toolkits and extensions. It is important to note that if this command is used to do this, on pre JS ROMs, the commands in the machine code Toolkit cannot be used in the same program which links them in. A typical boot program would therefore be:

```
100 a=RESPR (1024) : LBYTES flp1_Toolkit_ext,a : CALL a
110 LRUN flp1_Main_bas
```

Another use of LBYTES may be to load a screen which has been designed in a drawing program. A normal QL screen is a maximum of 32768 bytes long, however quite often screens can be much larger, so it is important to ensure that you check the length of the file before loading in what may be a screen file. To load a screen under SMS, use:

```
10 OPEN_IN #3,flp1_Loading_scr
20 scr_length=FLEN (#3)
30 scr_size=SCR_YLIM * SCR_LLEN
40 IF scr_size < scr_length
42 PRINT #0,'Screen resolution is too small for the saved file.':STOP
45 END IF
50 IF scr_size>scr_length:PRINT #0,'Screen resolution is too big for the saved␣
↪file.':STOP
60 LBYTES #3,SCR_BASE
70 CLOSE #3
```

**WARNING**

There is no check on the value of start, so ensure that you only try to LBYTES machine code into RAM which has been set aside with ALCHP or RESPR. Also ensure that the file is not too long to fit in the area of RAM allocated.

**CROSS-REFERENCE**

Normally code loaded with *LBYTES* has been saved using *SEXEC* or *SBYTES*. *FLEN* allows you to find out the length of a file, *FTYP* its file type.

## 19.6 LCM

| | |
|---|---|
| Syntax | LCM ($x^1$, $x^2$, $^*$[,$x^i$]$^*$) where $x^i$=0..INTMAX |
| Location | Math Package |

LCM is a function which takes two or more numeric parameters and finds their least common multiple, ie. the smallest number which can be divided by all of the parameters without a remainder. The parameters should be positive integers.

**Example**

```
PRINT LCM (2,3,4)
```

returns 12 and indeed 12/2=6, 12/3=4 and 12/4=3.

**NOTE**

If you are not looking for the least common multiple but any common multiple then simply multiply all the numbers, eg. 2*3*4=24.

**CROSS-REFERENCE**

*GCD*

## 19.7 LDRAW

| | |
|---|---|
| Syntax | LDRAW x1,y1 TO x2,y2, col |
| Location | HCO |

This command is similar to DRAW, ie. it draws quite a thick line on the screen, but LDRAW is part of the same Toolkit as SET and uses a different col parameter to DRAW, see SET for that.

**Example**

Well, the sample listing which follows on below, is a bit too long for a simple demonstration of LDRAW.

The variable rstep% in line 110 determines via pics% how much memory is required to run the animation.

Unless you have SMS or Minerva, you will have to replace i% and j% by i and j.

```
100 WINDOW 512,64,0,0: PAPER 0: INK 3: CLS
110 rstep% = 20
120 pics% = 360 / rstep%: DIM adr(pics%): i% = 0
130 FOR r = 0 TO 360-rstep% STEP rstep%
140   i%=i%+1: CLS: AT 0,0: PRINT "(";i%;")";TO 6;r;"ø"
150   PYRAMID 20, 256, 32, r, r, r, 3
160   adr(i%) = ALCHP(HEX("2000"))
170   IF NOT adr(i%) THEN CLCHP: STOP: REMark memory overflow
180   MM_MOVE HEX("20000"), adr(i%), HEX("2000")
190 END FOR r
200 REPeat Animation
210   FOR i% = 1 TO pics%
220     MM_MOVE adr(i%), HEX("20000"), HEX("2000")
230     IF KEYROW(1)&&8 THEN EXIT Animation
240   END FOR i%
250 END REPeat Animation
260 CLCHP: STOP
270 :
290 DEFine PROCedure PYRAMID (size, px%,py%, rotx,roty,rotz, c%)
300   LOCal i%, j%, p1(2), p2(2)
310   RESTORE 410
320   FOR i% = 1 TO 8
330     READ p1(0),p1(1),p1(2), p2(0),p2(1),p2(2)
340     ROTATION p1(0),p1(1),p1(2), rotx, roty, rotz
350     ROTATION p2(0),p2(1),p2(2), rotx, roty, rotz
360     FOR j%=0 TO 2: p1(j%)=size*p1(j%): p2(j%)=size*p2(j%)
370     LDRAW px%+p1(0),py%+p1(1) TO px%+p2(0),py%+p2(1), c%
380   END FOR i%
390   RETurn
```

(continues on next page)

(continued from previous page)

```
400 :
410   REMark base square
420   DATA -1, -1, 0, 1,-1, 0
430   DATA 1, -1, 0, 1, 1, 0
440   DATA 1, 1, 0, -1, 1, 0
450   DATA -1, 1, 0, -1,-1, 0
460   REMark top
470   DATA -1, -1, 0, 0, 0, 2
480   DATA 1, -1, 0, 0, 0, 2
490   DATA 1, 1, 0, 0, 0, 2
500   DATA -1, 1, 0, 0, 0, 2
510 END DEFine PYRAMID
520 :
530 :
540 DEFine PROCedure ROTATION (x, y, z, wx, wy, wz)
550   REMark rotate point (x,y,z) by angles wx, wy and wz
560   REMark in degrees around point (0,0,0)
570   LOCal x1, y1, x2, z2
580   LOCal cx, cy, cz, sx, sy, sz
590   cx = COS(RAD(wx)): cy = COS(RAD(wy)): cz = COS(RAD(wz))
600   sx = SIN(RAD(wx)): sy = SIN(RAD(wy)): sz = SIN(RAD(wz))
610   x1 = x * cz -y * sz
620   y1 = x * sz + y * cz
630   x = x1 * cy - z * sy
640   z2 = x1 * sy + z * cy
650   y = y1 * cx + z2 * sx
660   z = -y1 * sx + z2 * cx
670 END DEFine ROTATION
680 :
700 DEFine PROCedure MM_MOVE (addr1, addr2, bytes)
710   REMark move memory
720   LOCal routine
730   IF VER$ = "JSL1" THEN
740     routine = PEEK_W(344) + 16384
750     CALL routine, bytes, 2, 3, 4, 5, 6, 7, addr2, addr1
760   ELSE
770     REMark with HCO:
780     BMOVE addr1, addr1+bytes TO addr2
790   END IF
800 END DEFine MM_MOVE
```

**NOTE 1**

LDRAW assumes that the screen is in a resolution of 512x256 pixels and is located at $20000.

**NOTE 2**

LDRAW only works correctly in MODE 4.

**WARNINGS**

See SET.

**CROSS-REFERENCE**

*DRAW*. Please use *LDRAW* only if you know what you are doing, do not intend to write user-friendly programs, and especially if you do not intend to show your program listing to someone else! You can always use *LINE* and *LINE_R*, commands, *DOTLIN* and *XDRAW*, which can draw dotted lines (*DOTLIN*) or work in *XOR* mode (*XDRAW*).

## 19.8 LEFT

| | |
|---|---|
| Syntax | LEFT [ #channel ] |
| Location | QSOUND |

This command will move the text cursor left one column in the specified channel (default #1). If there is a pending newline on the specified channel (for example after a PRINT command) this will be cleared, making it as if the last PRINT (or INPUT) statement ended with a comma - for example:

```
100 PRINT 'Hello World'
110 PRINT 'THIS LINE IS PRINTED AFTER A PENDING NEWLINE'
120 LEFT
130 PRINT 'THIS OVERWRITES PART OF THE LAST TEXT'
```

'Out of Range' will be reported if you try to move the cursor left past column zero.

**CROSS-REFERENCE**

*AT* allows you to position the text cursor. *PRINT*, *TO*, *INPUT* and *CURSOR* also affect the text cursor.

## 19.9 LEN

| | |
|---|---|
| Syntax | LEN (string$) |
| Location | QL ROM |

The function LEN returns the number of characters contained in the given string expression. However, due to the QL's native coercion routines, the expression passed as a parameter need not be a string (!)

**Examples**

```
x=100: PRINT LEN(x): REMark Returns 3.
PRINT LEN ('A string'): REMark Returns 8.
DIM x$(12): PRINT LEN (x$): REMark Returns 0, but add the following
: x$='Hello': PRINT LEN(x$): REMark Returns 5, the same as PRINT x$(0)
```

**NOTE**

On pre-JS ROMs, if you use PRINT LEN(x$), an 'Out of Memory' error will be reported if you have previously tried to make x$ longer than 32766 characters, for example with:

```
x$=FILL$('x',32764)
x$=x$&'xxx'
PRINT LEN (x$)
```

**CROSS-REFERENCE**

*FILL$* returns a string of a specified length. *DIMN* returns important information about arrays. See also the Compatibility Appendix for some important information concerning string lengths.

## 19.10 LET

| | |
|---|---|
| Syntax | [LET] variable=expression |
| Location | QL ROM |

The command LET has only been implemented to make SuperBASIC more compatible with other versions of BASIC. It assigns a specific value to the specified variable, which can be of any type. The command may actually be omitted altogether. Normally any mistake in this command results in an 'Error in Expression' report.

**Examples**

```
LET x=100+10\*20
```

Assigns the value 300 to the variable x.

```
x=100+10\*20
```

Is exactly the same as above.

```
LET a$='Hello '&x
```

This places the string 'Hello 300' into the variable a$. The value of x is converted into a string and then appended.

```
LET position(100)=10
```

This assigns the value 10 to the 101st element of the array position (see DIM).

**NOTE 1**

On the AH ROM, you need to be careful of what is being assigned to a numerical variable: LET X=”.” did not produce an error on this ROM. Compare this with LET X='0.12' which in fact assigns the value 0.12 to the variable x due to coercion.

**NOTE 2**

It may be useful to explain the error codes which may be reported when trying to assign a value to a variable. Under SMS the improved interpreter will report more meaningful errors if you try to use this command incorrectly and therefore it is these errors which are highlighted.

**Assignment can only be to a variable or array element**

This is reported if you try to assign a value to a Procedure or Function name, eg: PRINT = 100

On other versions this causes an …

**Error in Expression**

When assigning values to arrays there are four possible error reports:

**Only arrays or strings may be indexed**
This will be reported if you try to assign a value to an undimensioned array, for example if you used the line: position (100)=10 without having used the line: DIM position (200) beforehand. On other implementations, this causes the error ...

**Bad Name**

**Cannot assign to sub-array**
We have not been able to find a situation when this error occurs.

**Unacceptable array index list**
This is reported normally if you try to use too many indices to reference an existing array, for example: DIM x(100) : PRINT x(10,10)

On other implementations this causes an

**Out of Range**

**Array index out of range**
This is reported if you try to use an index which is greater than that used when the array was dimensioned, for example: DIM x(100) : x(101)=100

On other implementations this also causes an

**Out of Range**

**WARNING**

On SMS, you can easily crash SBASIC by missing out an index on an assignment to a DIMensioned array, for example:

```
DIM x(100) x (10, ) = 100
```

Will report Not Complete:

```
x(10, , ) = 100
```

Will crash SBASIC.

On Minerva (and possibly other ROM versions) both of these merely report 'Error In Expression'.

**CROSS-REFERENCE**

*READ* and *INPUT* also allow you to assign a value to a variable.

## 19.11 LEVEL2

| | |
|---|---|
| Syntax | present = LEVEL2(#channel) |
| Location | DJToolkit 1.16 |

If the device that has the given channel opened to it has the level 2 drivers, then present will be set to 1, otherwise it will be set to 0. The level 2 drivers allow such things as sub_directories to be used, when a *DIR* is done on one of these devices, sub-directories show up as a filename with '->' at the end of the name. Gold Cards and later models of Trump cards have level 2 drivers. Microdrives don't.

**EXAMPLE**

```
2500 DEFine PROCedure MAKE_DIRECTORY
2510   LOCal d$, t$, l2_ok, ch
2520   INPUT 'Enter drive names :';d$
2530   IF d$(LEN(d$)) <> '_' THEN d$ = d$ & '_': END IF
2540   PRINT 'Please wait, checking ...'
2550   ch = DJ_OPEN_OVER (d$ & CHR$(0) & CHR$(0))
2560   IF ch < 0: PRINT 'Cannot open file on ' & d$ & ', error: ' & ch: RETurn
2570   l2_ok = LEVEL2(#ch)
2580   CLOSE #ch
2590   DELETE d$ & CHR$(0) & CHR$(0)
2600   IF l2_ok
2610     INPUT 'Enter directory name please : ';t$
2620     MAKE_DIR d$ & t$
2630   ELSE
2640     PRINT 'Sorry, no level 2 drivers!'
2650   END IF
2660 END DEFine MAKE_DIRECTORY
```

## 19.12 LGET

| | |
|---|---|
| Syntax | LGET [#ch\position,] [item $^*$[,item$^i$]$^*$ ..] or<br>LGET [#ch,] [item $^*$[,item$^i$]$^*$ ..] |
| Location | SMSQ/E |

This command is very similar to BGET, although this fetches a longword (4 bytes) at a time (in the range 0..$2^{32}$-1) from the given channel (default #3).

**NOTE**

LGET is affected by TRA.

**CROSS-REFERENCE**

See *BGET*. *LPUT* is complementary function. *WGET* allows you to fetch word values.

## 19.13 LINE

| | |
|---|---|
| Syntax | LINE [#chan,] [x,y] [TO $x^1$,$y^1$] $^*$[[;$x^i$,$y^i$] [TO $x^j$,$y^j$] ]$^*$ |
| Location | QL ROM |

This command is part of the QL's graphics repertoire and allows you to draw a straight line in the specified channel (default #1) in the current INK colour between any two points. As with all of the other graphics commands, the exact size and position of the line depends upon the current SCALE. Unfortunately, there is no way of making the line any thicker, other than by drawing parallel lines. Although the above syntax may seem rather complex, this can be explained as follows:

If the separator TO appears between any two sets of co-ordinates, then a line will be drawn between those two co-ordinates.

If however the two sets of co-ordinates are the same, nothing will be drawn, eg: LINE 10,10 TO 10,10 has no effect.

If the start co-ordinates are not specified, then the current graphics cursor is used as the one end of the line, eg: LINE 10,10 TO 15,10 TO 20,20 will draw a line between the points (10,10) and (15,10) and then a line between (15,10) and (20,20). The graphics cursor is placed at the last set of co-ordinates.

If the separator TO does not appear, then no line is drawn and the graphics cursor is moved to the last set of co-ordinates. For example: LINE 10,10 and LINE 20,20,10,10 have exactly the same effect - they both place the graphics cursor at the point (10,10).

Any part of the lines which lie outside of the specified channel will not be drawn, but no error will be reported.

**Example**

A simple demonstration program:

```
100 MODE 8 110 WINDOW 448,200,32,16:PAPER 0:CLS
120 SCALE 100,0,0
130 OVER -1
140 REPeat loop
150   xstep=RND
160   INK RND(7)
170   FOR i=1 TO 360 STEP xstep
180     ix=RAD(i)
190     LINE 50,50 TO 50+COS(ix)*50,50+SIN(ix)*50
200   END FOR i
210 END REPeat loop
```

**NOTE**

On a MG ROM, you may find that the last point is not always plotted.

**CROSS-REFERENCE**

*LINE_R* is very similar. See also *ELLIPSE*, *CIRCLE*, *ARC*, *POINT* and *SCALE*.

## 19.14 LINE_R

| | |
|---|---|
| Syntax | LINE_R [#chan,] [x,y] [TO $x^1$,$y^1$] *[[;$x^i$,$y^i$] [TO $x^j$,$y^j$] ]* |
| Location | QL ROM |

This command is very similar to LINE, except that all co-ordinates are taken to be relative to the current graphics cursor.

**CROSS-REFERENCE**

Please see *LINE*, *CIRCLE_R*, *ARC_R*, *ELLIPSE_R* and *POINT_R*.

## 19.15 LINKUP

| | |
|---|---|
| Syntax | LINKUP file$ |
| Location | Memory Toolkit (DIY Toolkit Vol H) |

This command is similar to LRESPR except that it will work even if jobs are running in the system. Although it loads the specified file into the common heap, it marks the area of memory as permanent and therefore this memory will not be removed by CLCHP or NEW. This therefore provides a safe means of linking in new toolkits and device drivers permanently even when Jobs are have already been EXECuted. Unlike LRESPR the default data device is not supported and the filename must be supplied in full as a string.

**CROSS-REFERENCE**

See *RESERVE* and *DISCARD*. Also see *LRESPR* and *ALCHP*.

## 19.16 LINT2

| | |
|---|---|
| Syntax | LINT2 [#ch] |
| Location | Beuletools |

This command lists all interrupt (level 2) service routines and their link pointers to the given channel (default #1). To understand this list, you will need to refer to documentation on the operating system (QDOS).

**CROSS-REFERENCE**

*LSCHD* and *LPOLL* list other information about the current system interrupts. Details of the external interrupt service list is contained in the QDOS/SMS Reference Manual Section 6.

## 19.17 LIST

| | |
|---|---|
| Syntax | LIST [#ch,] [range $^{*}$[,range$^{i}$]$^{*}$ ] |
| Location | QL ROM |

This command lists (in ASCII form) the specified range of the currently loaded SuperBASIC program to the specified channel (default #2). Range must be in the form: [[start_line] TO [end_line]].

The default start_line is 1 and the default end_line is 32767, therefore if no range is given, the LISTing range defaults to: 1 TO 32767.

Except under SMS, when the last line of the given range is reached, a table is set up which stores the current list range. This list range contains a list of the lines of the program which are currently shown in #2 - if you alter one of these lines (for example with EDIT or DLINE), then the listing in #2 is re-drawn to reflect the change. Alterations to lines outside the list range will have no effect.

Again, except under SMS, special note is also taken of the program line just above the displayed listing, and the program line just below the displayed listing - if either of these lines is altered, then the display will scroll accordingly to show the newly altered line on screen.

**Examples**

```
LIST #3
```

List the whole of the program in #3

```
LIST 1
```

List program line 1 in #2

```
LIST 100,1000 TO
```

List lines 100 and from 1000 onwards in #2

```
OPEN#3,SER1: LIST#3: CLOSE#3
```

will list the current program to a printer connected to ser1.

**NOTE 1**

Except under SMS, you may sometimes find a large chunk of the program listing scrolling before your eyes if you alter a line outside the range displayed in #2. This should not create any problems and generally occurs when you press Break before the List Range has been updated.

**NOTE 2**

Version 2.13 (and later) of Toolkit II alters this command so that if you are using LIST to output to a file, any errors will be reported (such as 'Device Full' or 'Not Complete').

**NOTE 3**

Prior to SMS v2.67 LIST #ch where #ch does not exist would attempt to SAVE the file.

**CROSS-REFERENCE**

When *LIST*ing to a file, this command is the same as *SAVE*. *DLINE*, *ED*, *EDIT*, and *RENUM* are other commands for dealing with a SuperBASIC program in memory.

## 19.18 LIST_TASKS

| | |
|---|---|
| Syntax | LIST_TASKS [#ch] |
| Location | TASKCMDS (DIY Toolkit Vol J) |

LIST_TASKS is nearly the same as JOBS, but the output is slightly different. Each line written to the specified channel (default #1) consists of the job name, job number, job tag and priority. A 'w' appended to the priority indicates that the job is currently suspended.

**CROSS-REFERENCE**

*JOBS* is similar.

## 19.19 LMAR

| | |
|---|---|
| Syntax | LMAR(n) with n=0..255 |
| Location | Beuletools |

This function returns the control codes needed to set the left margin to n characters on EPSON compatible printers: PRINT LMAR (10) is the same as PRINT CHR$(27)&'l'&CHR$(10)

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*, *DBL*, *ENL*, *PRO*, *SI*, *NRM*, *UNL*, *ALT*, *ESC*, *FF*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 19.20 LN

| | |
|---|---|
| Syntax | LN (x) |
| Location | QL ROM |

This function returns the natural logarithm of the given value (in base e), so that $e^{LN(x)}$=x. Due to the nature of power numbers, the range of x is 0>x<=$2^{2046}$.

Logarithms were first invented to make multiplication and division easier, because whatever base you are working in, multiplication and division can be calculated by using logarithms. For example, x*y is the same as EXP(LN(x)+LN(y)), or $10^{(LOG10(x)+LOG10(y))}$; and x/y is the same as EXP(LN(x)-LN(y)), and $10^{(LOG10(x)-LOG10(y))}$.

Another reason is that logarithms can make it easier to calculate powers, for example, $10^{(p*LOG10(y))}$ gives the same answer as $y^p$, for any value of y or p.

Another use for logarithms is to enable square roots to be calculated. On the assumption that x*x=$10^{(2*LOG10(x))}$, the square root of a number y can be calculated using the formula: $10^{(LOG10\ (y)\ /\ 2)}$.

Natural logarithms (base e) are generally used in theoretical mathematics, as this can be useful in differentiation, since if y=$e^x$, dy<dx<y. Because negative values of x cannot be handled by logarithms (in any base - this is because $x^y$ must always be greater than zero!), you will need to check for negative values and zero values separately.

**CROSS-REFERENCE**

*EXP* converts natural logarithms to their true numbers in base 10, *LOG10* provides logarithms in base 10 (common logarithms), and *LOG2* provides base 2 logarithms.

## 19.21 LOAD

| | |
|---|---|
| Syntax | LOAD device_filename or<br>LOAD [device_]filename (Toolkit II) |
| Location | QL ROM, Toolkit II |

This command looks for a SuperBASIC program held on the given device under the specified filename (a program file), reporting the error 'not found' if either the device or the filename does not exist. If found, it then clears any current SuperBASIC program out of memory, closes all channels with a channel number greater than #2, turns off any WHEN processing, and performs a CLS on #0, #1 and #2. Each line of the program file is loaded into memory and then parsed as if it had been entered into the command line by the user. If any lines cannot be parsed (ie. they would normally generate a 'bad line' error), then the word MISTake is inserted into the line after the line number and the loading process continues.

Under SMS when the program has been loaded, if there have been any errors in the program, the error 'MISTake in Program' is reported, or any other Interpreter error, with the line number listed.

Program files are stored on directory devices by the computer as pure ASCII files, allowing them to be imported into text editors for ease of editing (or even to be created in separate editing programs), copied direct to a printer (using the COPY_N command), and VIEWed on screen.

However, this means that the program has to be parsed each time that it is loaded, making the loading process quite slow. This can however be circumvented by using a fast loading utility - we highly recommend QLOAD from Liberation Software for this purpose.

If the program file contains some lines in it which do not have line numbers, then these are automatically executed as if they had been typed direct into the keyboard. For example, one method of software protection would be to turn off the Break key on loading and then RUN the program. This can be achieved by entering the following as direct commands, with the desired program in memory:

```
OPEN_NEW #3,flp1_file
LIST #3 PRINT #3,'BREAK_OFF':RUN'
CLOSE #3
```

This actually opens a new file, and inserts as direct commands BREAK_OFF and RUN after the body of the program (LIST in this instance is similar to SAVE except that it allows you to add further text to the end of the program file).

These two commands will be interpreted immediately that flp1_file has been loaded, thus preventing anyone from looking at the listing (the break key is disabled and the program immediately RUN).

Unfortunately though, this does not really work very well, as you cannot stop the user from VIEWing the file on screen!!

If you have Toolkit II present, then if a device is not specified, or LOAD cannot find the specified file on the given device, then Toolkit II will add the default data device to the filename. If the file still cannot be found, then the default program device is used instead.

**Example 1**

To load a file Test1_bas on mdv1_ (the default data device is flp1_ and the default program device is flp2_):

```
LOAD mdv1_Test1_bas
```

If Toolkit II is present and Test1_bas is not on mdv1_ (or there is not a microdrive cartridge in mdv1_), the default data device is added, equivalent to:

```
LOAD flp1_mdv1_Test1_bas
```

If the file is still not found, the default program device is used, which is equivalent to:

```
LOAD flp2_mdv1_Test1_bas
```

**Example 2**

Some examples showing the capabilities of LOAD:

```
LOAD 'n' & station & '_flp1_'&file$
```

Loads the given file from flp1_ on the given network station.

```
LOAD ser1c
```

Loads a file from the device attached to ser1.

```
LOAD neti_3
```

Loads a file which will be SAVEd over the network by station 3.

**NOTE 1**

LOAD can leave error trapping enabled on JS and MG ROMs - see WHEN ERRor for details.

**NOTE 2**

Minerva users will notice that in current versions, LOAD clears both screens even if #0, #1 and #2 are all on the same screen.

**NOTE 3**

LOAD allows programs which have been created on Minerva using integer tokenisation to be loaded into any other ROM without any problems - any numbers in the program file are automatically converted to floating point tokens (or long/short integers if integer tokenisation is enabled), thus preventing any problems.

**NOTE 4**

LOAD cannot be used from within a PROCedure or FuNction unless you have a JS ROM, MGx ROM, SMS or Minerva v1.83+. On other implementations, this causes the error 'Not Implemented'.

**NOTE 5**

Except under SMS, line numbers can be added to a numberless program file using AUTO - please refer to AUTO.

**NOTE 6**

On Minerva v1.86, LOAD could become confused when used inside a program.

**NOTE 7**

Since Toolkit II v2.22 (and on Minerva), LOAD will refuse to try and load a file unless its file type is 0 (see FTYP).

**NOTE 8**

Any commands which appear on the same line as LOAD (after the LOAD command) will be ignored.

**SMS NOTES**

LOAD has been re-written so that it will also load files saved with the QLOAD utility from Liberation Software (which is now part of SMS). If the specified filename does not end in _SAV or _BAS, then if the specified filename does not exist, before trying the default data device and the default program device (see above), LOAD will first of all try the filename with _BAS appended and if still not found, will try the filename with _SAV appended.

So if the default data device is flp1_ and the default program device is flp2_, LOAD ram1_TEST will look for the following files:

- ram1_TEST
- ram1_TEST_bas
- ram1_TEST_sav
- flp1_ram1_TEST

- flp1_ram1_TEST_bas
- flp1_ram1_TEST_sav
- flp2_ram1_TEST
- flp2_ram1_TEST_bas
- flp2_ram1_TEST_sav

Only if none of these filenames exist will it report a 'Not Found' error.

**CROSS-REFERENCE**

*SAVE* saves the current SuperBASIC program in memory. *LRUN* automatically runs the program after loading. *MERGE* and *MRUN* are similar commands. Also see *QLOAD* and *RELOAD*. *EXEC* allows you to load a multitasking program (normally a machine code program or a compiled program). *LBYTES* allows you to load a section of memory.

## 19.22 LOADPIC

| | |
|---|---|
| Syntax | LOADPIC file$ |
| Location | PICEXT |

This command will load an uncompressed 32K screen file and display it on the main screen. - This works exactly the same as LBYTES file$,131072. Note that LOADPIC needs the full filename to be supplied as a string.

**Example**

LOADPIC "flp1_Example_scr"

**NOTE 1**

LOADPIC assumes that the screen will be located at $20000 and will therefore not work on Minerva's second screen.

**NOTE 2**

LOADPIC will not work on high resolution screens as it expects the screen to be 512x256 pixels.

**CROSS-REFERENCE**

*SAVEPIC*, *SBYTES*, *LBYTES*, *SCREEN*, *EXPAND*, *COMPRESS*.

## 19.23 LOCal

| | |
|---|---|
| Syntax | LOCal $var^1$ $^*$[, $var^x$ [($index^1$ $^*$[$index^i$]$^*$ )] ]$^*$ |
| Location | QL ROM |

This command must only be used as the first executable line within either a PROCedure or FuNction definition block (ie. it can only be preceded by REMark lines) - if it is used elsewhere, it will cause a 'bad line' error when the program is RUN. Under SMS's improved interpreter the error 'Misplaced LOCal' will be reported.

LOCal must be followed by a list of variables which are said to be 'local' to that definition block. This means that although a variable may already have been used within the main body of the program, if it is local to that definition block, on entry its value is stored and it is then made 'unset' (without value), and can then be used for any means within that definition block (or within any sub-procedure or sub-function called by that definition block).

When the definition block is left (with END DEFine or RETurn), the variable is restored to its original value.

Arrays can also be made LOCal by placing an index after their name, which is used to specify their size (as with DIM). Indeed this is the only way in which a simple variable can also be used as an array. In any event, the parameters contained in the definition line are local to that definition block and can also be safely used in the main program - these are in fact swapped with the actual parameters passed for the duration of the definition block (see DEFine PROCedure).

**Example**

This program shows the status of three variables at various stages - note how x can be used as an array in the main program and a simple variable within the PROCedure definition block:

```
100 DIM x(10)
110 test$='Wait'
120 moder=4:x(1)=10
130 PRINT moder,test$,x(1)
140 Change_vars
150 PRINT moder,test$,x(1)
155 :
160 DEFine PROCedure Change_vars
170   LOCal moder(2,10),x,test$
180   PRINT moder(1,5),test$,x
190   test$='Changed':moder(1,5)=10
200   x=5
210   PRINT moder(1,5),test$,x
220 END DEFine
```

This produces the following output:

```
4 Wait 10 line 130
0 * * line 180, local variables
10 Changed 5 line 210, local variables
4 Wait 10 line 150
```

**NOTE 1**

On pre MG ROMs, any more than nine parameters may corrupt the program, by replacing names with PRINT towards the end of a program. This can however be circumvented by increasing the size of the Name Table by 8 bytes for each name (plus a little more for luck), by using the line:

```
CALL PEEK_W(282)+36,N
```

This bug is fixed on the ST/QL Emulator (with E-Init software v1.27+), Minerva and SMS.

**NOTE 2**

On most ROMs, you cannot LOCal the names of the parameters passed to the PROCedure or FuNction.

ROMs which can cope with this will simply set the passed value to undefined. Type in the following small procedure test:

```
100 DEFine PROCedure test(a,b)
110   LOCal a
120   PRINT a,b
130 END DEFine
```

If your interpreter behaves correctly then:

```
test 3,2
```

will write:

```
* 2
```

SMS will print:

```
0 2
```

Any reference to a in the procedure, eg. a=a+1, will break with an error in expression (-17) because the LOCal declaration of a undefined the passed parameter. You would need to expressly assign a value to a within the PROCedure for this to work. This works correctly on Minerva ROMs (ie. a is unset by the LOCal command).

**CROSS-REFERENCE**

*DIM* sets up arrays normally. *DEFine PROCedure*, *DEFine FuNction* and *END DEFine* are used to identify definition blocks.

## 19.24 LOCK

| | |
|---|---|
| Syntax | LOCK file,code$,code |
| Location | CRYPTAGE |

This command encodes the given file (the full filename must be stated) using two codes, a string and a number, for security. Code$ can be any string and the code number (an integer) must range between 0 and 32767. Decoding with UNLOCK is only possible if both codes are known, so do not forget them otherwise the file will be lost.

**Example**

LOCK ram1_secret_txt,"Phew",7241

**CROSS-REFERENCE**

*UNLOCK* has the same syntax as *LOCK* but deciphers *LOCK*ed files.

## 19.25 LOG2

| | |
|---|---|
| Syntax | LOG2 (x) |
| Location | Math Package |

This function returns the logarithm to the base 2 of the given number, which is calculated as LN(x)/LN(2).

**Example**

The greatest number which can be handled by SuperBASIC is returned by INF as 1.61585E616. This is exactly $2^{2047}$, because LOG2(INF)=2047 (ie. $x=2^{LOG2(x)}$).

**CROSS-REFERENCE**

*LOG10*, *LN*, *INF*.

## 19.26 LOG10

| | |
|---|---|
| Syntax | LOG10 (x) |
| Location | QL ROM |

The function LOG10 calculates the logarithm to the base 10 of the given number. For the non-mathematicians out there: $x=10^{LOG10(x)}$.

**Examples**

```
100 INPUT "Integer Number:"!x
110 PRINT "This number has"!INT(1+LOG10(ABS(x)))!"digits."
```

The trivial function LOGN finds the logarithm of x to any base b which makes sense:

```
10 DEFine FuNction LOGN (x,b)
20   RETurn LN(x)/LN(b)
30 END DEFine LOGN
```

**CROSS-REFERENCE**

*LN*, *LOG2*.

## 19.27 LOOKUP%

| | |
|---|---|
| Syntax | LOOKUP% (search$) |
| Location | Function (DIY Toolkit - Vol R) |

This function expects you to pass a string parameter which contains a name used by the SuperBASIC interpreter. This name can be a machine code Procedure or Function (such as are described here in this manual), or a SuperBASIC variable, PROCedure or FuNction. If the specified name is recognised then LOOKUP% returns the number of its entry in the name list. If the name is not recognised, then the value -7 is returned.

**Examples**

```
PRINT LOOKUP% ('PRINT')
```

will return 0 on most QL ROMs as this is normally the first name in the name list.

```
PRINT LOOKUP% ('FSERVE')
```

can be used to see if Toolkit II's fileserver is available.

**NOTE 1**

This function will only look at the name list for SuperBASIC Job 0, so although it can be used from within a compiled task to look at Job 0, it cannot be used to look at a multiple BASIC interpreter!!

**NOTE 2**

This function will only work correctly with machine code Procedures and Functions on SMS.

**CROSS-REFERENCE**

See*ELIS*, *KEY_ADD*. *_NAME$* allows you to look at the name list. See also *FLIS* and *FIND*.

## 19.28 LOWER$

| | |
|---|---|
| Syntax | LOWER$ (string$) |
| Location | Function (DIY Toolkit - Vol R) |

This function takes the given string and converts any upper case letters to lower case and then returns the whole string. This will convert both UK and accented characters, although you may have to modify the source code to enable it to work with some international character sets.

**CROSS-REFERENCE**

Compare *UPPER$*. See also *ConvCASE$*.

## 19.29 LPOLL

| | |
|---|---|
| Syntax | LPOLL [#ch] |
| Location | Beuletools |

This command lists all polling interrupts and their link pointers to the given channel (default #1). While this text was being written, LPOLL produced the following list:

```
List of polled tasks:
Link Pointer   Routine
1.   $0002B5D8 $000C1434
2.   $0002B8B8 $0009E0C2
3.   $0002CAAA $000BD056
4.   $0002B840 $0009E988
```

To understand these numbers, a deep knowledge of assembly language and the operating system is necessary. Generally, each interrupt is a kind of background job which does not appear in the job list. For further information, refer to system documentation.

**CROSS-REFERENCE**

*LSCHD* and *LINT2* list other internal routines which are running in the interrupts. *JOBS* lists all jobs.

## 19.30 LPR_USE

| | |
|---|---|
| Syntax | LPR_USE device |
| Location | Beuletools |

LPR_USE sets the default output device for LPRINT and LLIST. This can be any valid QDOS device name, provided it is not longer than 24 characters. The definition can be found with LPRINT$, the default is SER1 (ie. if LPR_USE has not yet been used).

**Examples**

```
LPR_USE par
LPR_USE ram1_print_dat
LPR_USE n2_ser1
LPR_USE con
```

**NOTE**

LPR_USE does not check the validity of the given device, so even completely wrong parameters are accepted:

```
LPR_USE #2
```

will set LPRINT$ to "2", LPRINT$ and LLIST will report the error.

**CROSS-REFERENCE**

LLIST, LPRINT$.

## 19.31 LPUT

| | |
|---|---|
| Syntax | LPUT [#ch\position,] [item *[,item$^i$]* ..] or<br>LPUT [#ch,] [item *[,item$^i$]* ..] |
| Location | SMSQ/E |

This command is the complement to LGET, in that it places the longword value for each item into the specified channel (default #3) at the current file position (or the specified position if the first variant is used).

**NOTE**

LPUT is affected by TRA.

**CROSS-REFERENCE**

See *BPUT* and *LGET*. *WPUT* and *PUT* are also similar.

## 19.32 LRESFAST

| | |
|---|---|
| Syntax | LRESFAST mc_file |
| Location | ATARI_REXT for QVME (v2.31+) |

This command is the same as LRESPR except that it will only work on a file in RAM disk and loads that file into FastRAM on the Atari TT.

**CROSS-REFERENCE**

See *LRESPR* and also*RESFAST*, *FREE_FAST*.

Compare*RESPR*, *ALCHP* and *FREE_MEM*.

## 19.33 LRESPR

| | |
|---|---|
| Syntax | LRESPR mc_file |
| Location | Toolkit II, THOR XVI |

This command is used as a quick way of loading and starting machine code routines (mainly Toolkits). It will grab enough memory from the Resident Procedure Area to hold the given file, load the file into memory and then call it. Toolkit II sub-directories and the default data device are supported. LRESPR could be re-written as the following SuperBASIC procedure:

```
100 DEFine PROCedure LRESPR (mc_file$)
110   LOCal length,adress
120   length=FLEN(\mc_file$)
130   adress=RESPR(length)
140   LBYTES mc_file,adress
150   CALL adress
160 END DEFine LRESPR
```

**Examples**

```
LRESPR BeuleTools_bin
LRESPR ram1_MyTool_obj
```

**NOTE 1**

It is impossible to remove a program loaded with LRESPR so that the occupied memory can be given back for other purposes.

**NOTE 2**

On version 2.23 (or later) of Toolkit II, LRESPR works even if a job is running because in this case, it will load the file into the Common Heap. CLCHP, NEW, CLEAR etc. do not remove code loaded in this way, so a crash is impossible.

**NOTE 3**

When using LRESPR (or any other means) to link in extensions to SuperBASIC, bear in mind that pre JS ROMs needed the command NEW (or LOAD / LRUN) before those commands will be available. This happens on MG ROMs sometimes as well.

**NOTE 4**

If this command is used to link a toolkit into a MultiBASIC under Minerva or a multiple SBASIC under SMS, then that toolkit will be local to that BASIC interpreter - when you remove that BASIC, the toolkit will also disappear.

**CROSS-REFERENCE**

See the second example for *ALCHP*. See also *LINKUP* and *LRESFAST*.

## 19.34 LRUN

| | |
|---|---|
| Syntax | LRUN device_filename or<br>LRUN [device_]filename (Toolkit II) |
| Location | QL ROM, Toolkit II |

This command is exactly the same as LOAD except for the fact that the program is automatically RUN as soon as loading is complete.

**CROSS-REFERENCE**

See *LOAD*!

## 19.35 LSCHD

| | |
|---|---|
| Syntax | LSCHD [#ch] |
| Location | Beuletools |

This command lists all scheduler loop tasks with their linked pointers to the specified channel (default #1). While this text was being written, the following list was produced:

```
List of scheduler loop tasks:
link pointer routine
1. $0002B848 $0009E9C0
2. $0002D140 $000ACC2A
3. $0002C0F0 $000B685C
4. $0002B648 $000C1572
5. $000B3964 $000AFAEE
6. $000B5FDA $000B50FE
7. $00001206 $0000120E
8. $00002D7C $00002D90
9. $00003504 $0000350C
```

An in-depth knowledge of the operating system and machine code is necessary to understand this list. Please refer to the operating system documentation.

**CROSS-REFERENCE**

*LPOLL*, *LINT2*.

## 19.36 LWC$

| | |
|---|---|
| Syntax | LWC$ (string$) |
| Location | LWCUPC |

The function exchanges all upper case characters in the given string to lower case characters and returns the result. Only the standard alphabet is recognised - umlauts etc. are ignored.

**CROSS-REFERENCE**

*UPC$* and *UPPER$* return the string in upper case characters.

CHAPTER

TWENTY

# KEYWORDS M

## 20.1 MACHINE

| | |
|---|---|
| Syntax | MACHINE |
| Location | SMSQ/E, SMSQ/E for QPC |

This function returns a value corresponding to the type of system on which SuperBASIC is running. The values currently returned are:

| MACHINE | Machine Type |
|---|---|
| 0 | ATARI ST / STM / STF / STFM |
| 1 | ORDINARY STE (1040) - NOT SUPPORTED!!! |
| 2 | MEGA ST or ST / STM / STF / STFM with REAL-TIME CLOCK |
| 4 | ATARI STACY |
| 6 | ATARI STE |
| 8 | MEGA STE |
| 10 | GOLD CARD |
| 12 | SUPER GOLD CARD |
| 16 | FALCON |
| 17 | Q40/Q60 |
| 18 | Q68 |
| 20 | SMSQmulator |
| 24 | ATARI TT 030 |
| 26 | Q-emulator |
| 28 | QXL |
| 30 | QPC |

On Standard QLs, MACHINE returns the above value plus 1 if HERMES is fitted.

On ATARI Computers, MACHINE returns the above value plus 1 is a BLITTER CHIP is fitted.

Users without SMSQ/E can use the command:

```
PRINT PEEK (SYS_VARS+HEX('a7')) && BIN ('0011111')
```

This should return the same values (except that 0 will also be returned on a standard QL without any expansion board or a THOR computer).

You can test for a THOR by using:

```
PRINT PEEK (SYS_VARS+HEX('84'))
```

which will be 0 on any other implementation. We do not know what it returns on the THOR range of computers.

**Example**

A program to test for the actual machine on which the program is running, as well as other interesting data:

```
100 v$=VER$:q$=QDOS$
110 processor$='00':FPU=0:em_type=0
120 IF v$<>'HBA'
130   m_type=PEEK(SYS_BASE+HEX('A7'))
140   em_type=m_type && BIN('11100000')
150   m_type=m_type && BIN('00011111')
160   SELect ON em_type
170     =1:em_type=3
180     =2:em_type=1
190     =4:em_type=2
200   END SELect
210   IF v$='JSL1'
220     PRINT 'MINERVA FITTED'
230     m_type=100 : processor$='08'
240   ELSE
250     IF m_type=0
260       IF PEEK (SYS_BASE+HEX('84'))<>0
270         IF q$(1)='4':PRINT 'THOR 1 Computer';
280         IF q$(1)='5':PRINT 'THOR 20 Computer';
290         IF q$(1)='6':PRINT 'THOR XVI Computer';
300         IF q$(1) INSTR '456'=0:PRINT 'UNKNOWN THOR Computer'
310         PRINT ' v';q$:STOP
320       END IF
330     END IF
340   END IF
350 ELSE
360   m_type=MACHINE:processor$=PROCESSOR
370   FPU=processor$(2):processor$=processor$(1)&'0'
380   em_type=DISP_TYPE
390 END IF
400 extra_chip=m_type MOD 2
410 m_type=(m_type DIV 2)*2
420 SELect ON m_type
430   =0: IF v$='HBA'
440         PRINT 'ATARI ST / STM / STF / STFM';
450         ELSE :PRINT 'STANDARD QL - ROM VERSION ';v$ : processor$='08'
460       END IF
470   =2: PRINT 'MEGA ST or ST / STM / STF / STFM with REAL-TIME CLOCK';
480   =4: PRINT 'ATARI STACY';
490   =6: PRINT 'ATARI STE';
500   =8: PRINT 'MEGA STE';
```

(continues on next page)

(continued from previous page)

```
510   =10: PRINT 'GOLD CARD';
520   =12: PRINT 'SUPER GOLD CARD';
530   =16: PRINT 'FALCON';
532   =17: PRINT 'Q40/Q60';
534   =18: PRINT 'Q68';
536   =20: PRINT 'SMSQmulator';
540   =24: PRINT 'ATARI TT 030';
542   =26: PRINT 'Q-emulator';
550   =28: PRINT 'QXL'
560   =30: PRINT 'QPC'
570 END SELect
580 SELect ON extra_chip
590   =0: PRINT
600   =1:
610     SELect ON m_type
620       =0,2,4,6,8,16,24:PRINT ' with BLITTER'
630         SELect ON em_type
640           =0: PRINT 'Original QL Emulator FITTED'
650           =1: PRINT 'Extended Mode 4 Emulator FITTED'
660           =2: PRINT 'QVME Emulator FITTED'
670           =3: PRINT 'Monochrome Display Only'
680         END SELect
690     =REMAINDER :PRINT ' with HERMES'
700     END SELect
710 END SELect
720 PRINT 'ROM VERSION - ';v$
730 PRINT 'OS VERSION - ';q$
735 IF m_type<>30
740   PRINT 'PROCESSOR - 680';processor$;
745 ELSE
750   PRINT 'INTEL PROCESSOR';
755 END IF
760 SELect ON FPU
765   =0 : PRINT
770   =1 : PRINT 'with Internal MMU'
780   =2 : PRINT 'with 68851 MMU'
790   =4 : PRINT 'with Internal FPU'
800   =8 : PRINT 'with 68881 or 68882 FPU'
810 END SELect
```

**NOTE 1**

The machine type was only stored with on the ST/QL on Level E-20 drivers onwards. E-Init v1.07+ is also required.

**NOTE 1**

The wrong value was returned for the Mega ST prior to v1.33 of E-Init.

**CROSS-REFERENCE**

See *PROCESSOR*, *DISP_TYPE*, *VER$* and *QDOS$*. *VER$*(1) and *SYS_BASE* replace *SYS_VARS*. See also *A_MACHINE*.

## 20.2 MAKE_DIR

| | |
|---|---|
| Syntax | MAKE_DIR subdirectory |
| Location | Level-2 and Level-3 Drivers, THOR XVI |

The command MAKE_DIR creates a sub-directory which allows a group of files to be regarded as one unit when the contents of a medium are listed. Operations other than the standard DIR, WDIR, WSTAT and WDEL, are not affected. A file belongs to a sub-directory if the sub-directory's name appears as a prefix of the file, whether the file was created before or after the sub-directory.

Sub-directories are only supported on Level-2 (or higher) floppy disks, hard disks and ramdisks.

A sub-directory name can be any name but any underscore at its end will be ignored by MAKE_DIR.

If there is no medium in the given device, or if you do not specify a device name, the current default data device will be used (see DATAD$).

Sub-directories are identified by a "->" in directory listings and programs can identify them by examining their file type (255).

Empty sub-directories can be deleted as normal files, but a sub-directory which is not empty, cannot be deleted: error -9 (in use) will be reported. Actually, sub-directories are normal files which hold a list of the files which are contained within them. The list consists of the standard file headers of these files, each being 64 bytes long.

A sub-directory file never decreases in length if a file in the sub-directory has been deleted, instead, the file header is just marked as deleted by setting the name to a zero string.

All normal operations are allowed on sub-directory files (except RENAME), and they can be accessed using OPEN_DIR and FOP_DIR.

**Example**

If FLP1_ contains an empty disk and we then save the current SuperBASIC program as flp1_my_prog_bas and a backup as flp1_backup_bas:

```
SAVE flp1_my_prog_bas
SAVE flp1_backup_bas
```

a directory of flp1_, produced with WDIR flp1_ (for example), shows the following contents:

```
my_prog_bas
backup_bas
```

Now, we create the sub-directory my (or my_) with:

```
MAKE_DIR flp1_my_
```

or:

```
MAKE_DIR flp1_my
```

(both forms are equivalent) and look at the directory again:

```
my ->
backup_bas
```

MAKE_DIR created a new file "my" (not "my ->") which is marked as a sub-directory with the "->" sign by the DIR, WDIR and WSTAT commands. The file type of my is 255, PRINT FTYP(\flp1_my) will return that.

But where is my_prog_bas? It was moved into the sub-directory my_ and DIR flp1_my_ will show it again.

**NOTE 1**

The QUBIDE interface does not allow you to use MAKE_DIR to create a sub-directory if any files already exist which would fall into that sub-directory.

**NOTE 2**

The level-2 device drivers introduced a new standard for subdirectories - other methods which were implemented in the past are not recognised in this (e)book.

**NOTE 3**

If a disk with a sub-directory is read by a level-1 device driver, the sub-directory appears as just another file and files which have been added to a sub-directory after its creation cannot be accessed or seen by the system. However, if a file had been created before the sub-directory, the level-1 device driver finds this file just as if the sub-directory did not exist. So, if you prepare a disk which should also be readable on level-1 device drivers, either don't use sub-directories or create them after everything else.

**NOTE 4**

Sub-directory names longer than 27 characters on Toolkit II may hang up the SuperBASIC interpreter. Since the system does not treat nested sub-directories differently, the above warning applies to long sub-directory prefixes as well. However, this lock up will only occur when creating sub-directories not when using them. This problem is fixed on SMS v2.85.

**NOTE 5**

A filename cannot be longer than 36 characters and as described above, sub-directories are prefixes which reduce the maximum possible length of a filename. If you try to create a file (eg. SAVE) in a sub-directory so that the combined length of the file name and sub-directory are longer than 36 characters, a 'not found' error will be returned.

**WARNING 1**

It is possible to create a sub-directory so that it cannot be removed any more **(do not try this on a hard disk, you have been warned)**

```
SAVE test_
MAKE_DIR test
```

The file test_ (with an underscore) has been moved into the test directory, but it cannot be deleted to empty test. - This has been fixed in drivers later than version 2.28.

**WARNING 2**

```
MAKE_DIR net_
MAKE_DIR "net"
```

and similar commands lock-up the machine, so if you want you create a sub-directory called 'net' in the current directory, use:

```
MAKE_DIR DATAD$ & "net".
```

**WARNING 3**

```
MAKE_DIR flp1__
MAKE_DIR flp1___
```

and similar commands could create recursive directories until this was fixed in SMS v2.77.

**CROSS-REFERENCE**

*FMAKE_DIR* is a syntactical variation of *MAKE_DIR*. *OPEN_DIR* and *FOP_DIR* allow you to read directories of disks as well as sub-directories on level-2 drivers. The *DUP*, *DDOWN*, *DNEXT* and *DATA_USE* commands are used to move around in a sub-directory tree. See *FOP_DIR* for a program which lists a sub-directory tree. To enable programs to read sub-directories which have not been written for that purpose, the DEV_ device exists (see *DEV_USE*). The only legal way of identifying a sub-directory is by examining its file type as returned by *FTYP* or *FILE_TYPE* for example.

## 20.3 MATADD

| | |
|---|---|
| Syntax | MATADD sum,matrix1,matrix2 |
| Location | Math package |

The command MATADD adds the two matrices contained in the arrays matrix1 and matrix2, setting the result in the array sum. The parameters, matrix1, matrix2 and sum, must all be arrays of the same dimensions, the same size and the same type. They can be of any number type, viz. floating point or integer (% suffix), but not string and (we must stress this point) floating point and integer arguments must not be mixed. If these conditions are not satisfied, then MATADD will break with a 'bad parameter' error (-15). Provided that the parameters follow this rule, the command MATADD sets all of the elements of the sum array to the sum of the respective elements of the two other arrays, matrix1 and matrix2.

**Example**

```
100 DIM a%(10,10,80), b%(10,10,80), c%(10,10,80)
110 MATRND a%,-5 TO 5: MATSEQ b%
120 MATADD c%,a%,b%
```

**CROSS-REFERENCE**

If you run this short example program (8000 internal loops!), you will notice the extraordinary speed of *MATADD* which is representative of the other MAT... functions; *MATSUB* is almost equivalent to *MATADD*.

## 20.4 MATCOUNT

| | |
|---|---|
| Syntax | MATCOUNT (array, value) or<br>MATCOUNT (array1, array2) |
| Location | Math Package |

MATCOUNT is a function which counts how often a certain value appears in the given array where array and value can be of any type (even strings) as long as they are of the same type. The second syntax allows you to pass two arrays array1 and array2 of the same type and dimensions, MATCOUNT will then compare these two arrays and return the number of different elements.

**Example**

The following programs compares two random integer arrays and will always print something around 33%:

```
100 DIM x%(1000), y%(1000)
110 MATRND x%,2: MATRND y%,2
120 PRINT MATCOUNT(x%,y%)/10;"%"
```

**CROSS-REFERENCE**

*MATCOUNT* comparisons are exact in that two numbers a and b are only regarded as equal if a=b in SuperBASIC terms. The same is true for strings, it means that their comparison is case-sensitive. *MATCOUNT1* differs from *MATCOUNT* (see below) only in the fact that comparisons are based on the SuperBASIC operator == instead of =.

## 20.5 MATCOUNT1

| | |
|---|---|
| Syntax | MATCOUNT1 (array, value) or<br>MATCOUNT1 (array1, array2) |
| Location | Math Package |

The function MATCOUNT1 is just a variation of MATCOUNT which performs comparisons not as exact as MATCOUNT. Numbers must only be almost equal, the absolute difference must be smaller than the absolute of the second number divided by 1E7: ABS (a-b) < ABS (b / 1E7). This is the case if a==b. MATCOUNT1 is therefore the same as MATCOUNT if integers are being dealt with. Comparison of strings is not case-sensitive, again this is analogous to the == operator: “QDOS”==”Qdos” is true while “QDOS”=”Qdos” is not.

**CROSS-REFERENCE**

*MATCOUNT*, *MATEQU*.

## 20.6 MATEQU

| | |
|---|---|
| Syntax | MATEQU array1, {array2 \| value} |
| Location | Math Package |

The command MATEQU sets up array1 in two different ways depending on the type of the second parameter: (1) If another array array2 of the same dimensions is supplied then each element of array1 is set to the corresponding element of array2; or (2) If the second parameter is not an array but a constant, variable or expression then each element of array1 is set to the given value. Array1, array2 and value can be of any type: integer, floating point or string. array1 and array2 must however be of the same type and have the same number of dimensions.

**Examples**

```
DIM a$(4,8), a%(2,2,2,2,2), a(0), b$(4,8)
MATEQU a$,"Hi there"
MATEQU a%,6
MATEQU a%,-PI
test$=9.5: MATEQU a%,test$
MATEQU a,9.5
MATEQU a$,b$
```

**NOTE**

Supercharge and Turbo users… sorry!

**CROSS-REFERENCE**

*MATRND*, *MATIDN*

## 20.7 MATDEV

| | |
|---|---|
| Syntax | MATDEV array[%] |
| Location | Math Package |

This function takes any numeric array and calculates a number from its values which gives information about their standard deviation.

**Example**

```
10 DIM x(10)
20 PRINT MATDEV (x)
```

gives 0 because all elements of x are equal and therefore, have no deviation. Add the line:

```
15 MATRND x,10
```

and the result will be be around 3.2.

**CROSS-REFERENCE**

*MATMEAN*

## 20.8 MATIDN

| | |
|---|---|
| Syntax | MATIDN matrix |
| Location | Math Package |

This command forces the square numeric array matrix to be initialised so that the matrix is given the algebraic identity for matrices of that size. This gives the matrix the following format:

```
1 0 0 . . . 0 0 0
0 1 0 . . . 0 0 0
0 0 1 . . . 0 0 0
```

(continues on next page)

(continued from previous page)

```
. . .       . . .
. . .       . . .
. . .       . . .
0 0 0 . . . 1 0 0
0 0 0 . . . 0 1 0
0 0 0 . . . 0 0 1
```

All elements on the diagonal line from the top left corner to the bottom right corner are set to 1 and all other elements are set to 0. This forms the identity matrix, which means that when a matrix of the same size is multiplied by this, the resultant matrix is the same as the original matrix, ie. matrix1 * matrix = matrix1.

**CROSS-REFERENCE**

*MATMULT* multiplies matrices.

## 20.9 MATINPUT

| | |
|---|---|
| Syntax | MATINPUT array [{\\|,|;|!}] |
| Location | Math Package |

The command MATINPUT reads each element of an array in turn from #1, so that you have to type them all in. The modifiers ';' and '!' place the cursor behind the last entry whilst ',' moves it to the next tab position. The default is '\' which forces a new line between entries - the '\' can be omitted.

**Example**

```
100 DIM a(1,2)
110 MATINPUT a,
```

**CROSS-REFERENCE**

*MATREAD*, *MATRND*, *FOR*

## 20.10 MATINV

| | |
|---|---|
| Syntax | MATINV matrix2,matrix1 |
| Location | Math Package |

The command MATINV takes the array matrix1, inverts it and stores the result in matrix2.

Inverting is a mathematical term and produces a result from a matrix which is similar to finding the reciprocal of a number, namely, the relation is expressed by the fact that the product of a number and its reciprocal is one and the product of a matrix and its inverse matrix is the identity matrix:

```
n=10: DIM A(n,n), B(n,n), C(n,n)
MATRND A
```

A is a random matrix.

```
MATINV A,B
```

makes B the inverted matrix of A.

```
MATMULT C,A,B
```

Multiply A with B and store the result in C. C will be almost identical to the matrix ONE defined with:

```
DIM ONE(n,n): MATIDN ONE
```

C and ONE do not have exactly the same values because of the limited precision of the QL maths package. Two conditions are absolutely necessary for MATINV to work:

```
- DET (matrix1) <> 0
- matrix1 and matrix2 must be square matrices
```

**Example**

A matrix A and an array b form a so-called "linear equation system" which has a solution x which is an array like b. This example will find the solutions x(i) of the system, for any positive value of n (the size of the matrix):

```
100 n=5
110 DIM A(n,n), AINV(n,n), b(n), x(n)
120 MATRND A: MATRND b
130 :
140 MATINV A,AINV
150 MATSCALM AINV,b TO x
160 PRINT "Solutions:"\x
170 IF ABS(DET)<1E-6 THEN PRINT "(dubious results)"
180 :
190 DEFine PROCedure MATSCALM (matrix,array1,array2)
200   LOCal i,j
210   FOR i=0 TO DIMN(matrix,1)
220     array2(i)=0
230     FOR j=0 TO DIMN(matrix,2)
240       array2(i)=array2(i)+array1(j)\*matrix(i,j)
250     END FOR j
260   END FOR i
270 END DEFine MATSCALM
```

The method of solving a linear equation system by calculating the inverted matrix is known as Cramer's Rule. The advantage is that if the matrix A is constant and only the array b varies for other situations, MATINV needs only be called once and not afterwards for each value of the array b.

**NOTE**

Calculation time takes longer as the size of the matrix increases eg. the above example will take nearly an hour to calculate n=100. MATINV cannot be stopped with <CTRL><SPACE> whilst number crunching.

**CROSS-REFERENCE**

It is highly recommended to check if *DET* is very close to zero after *MATINV* has been executed, if this is the case, *MATINV* may have found a result which does not exist:

```
IF ABS(DET) < 1E-6 THEN PRINT "dubious result"
```

This works because *MATINV* calls *DET* internally.

## 20.11 MATMAX

| | |
|---|---|
| Syntax | MATMAX (array[%]) |
| Location | Math Package |

This function finds the largest value contained in an integer or floating point array.

**NOTE**

This cannot be compiled with Supercharge or Turbo.

**WARNING**

A string array makes MATMAX hang the system.

**CROSS-REFERENCE**

*MATMIN* is the complementary function. See also *MAXIMUM* and *MAXIMUM%*.

## 20.12 MATMEAN

| | |
|---|---|
| Syntax | MATMEAN (array[%]) |
| Location | Math Package |

This function returns the average of the array's elements, calculated by the sum of the elements divided by the number of elements.

**NOTE**

Don't compile with Supercharge or Turbo.

**WARNING**

Avoid string parameters!

**CROSS-REFERENCE**

See *MATSUM* for an example.

## 20.13 MATMIN

| | |
|---|---|
| Syntax | MATMIN (array[%]) |
| Location | Math Package |

This function finds the smallest element in an integer or floating point array.

**NOTE**

Cannot be compiled with Supercharge or Turbo.

**WARNING**

A string array makes MATMIN hang the system.

**CROSS-REFERENCE**

*MATMAX* is the opposite function. Refer also to *MINIMUM* and *MINIMUM%* which are even quicker.

## 20.14 MATMULT

| | |
|---|---|
| Syntax | MATMULT product, matrix1, matrix2 |
| Location | Math Package |

The command MATMULT performs multiplication on matrices of floating point type. The matrix1 is multiplied with matrix2 and the result stored in product. Since a n x m matrix represents a linear transformation which takes n-dimensional vectors and produces m-dimensional vectors from them, the following conditions must be satisfied by the three matrices supplied to MATMULT:

- All matrices must be two-dimensional.
- DIMN (matrix1, 2) = DIMN (matrix2, 1)
- DIMN (matrix1, 1) = DIMN (product, 1)
- DIMN (matrix2, 2) = DIMN (product, 2)

The latter three conditions are obviously satisfied by square matrices.

**Example**

Multiplication of two matrices means that their effect on a vector is combined into one matrix. The following program demonstrates this on a simple square.

The square x is a list of four vectors. x is first rotated with ROT by 45°, the rotated square is stored in y.

Now this y is squeezed in size by one half with SQZ and stored in z. Lines 240 to 280 perform all this and show the process.

After a keystroke, the matrix ROTSQZ will be created as the product of ROT and SQZ. Again the original square is transformed but this time by ROTSQZ which rotates and squeezes in one go. This is done by lines 300 to 350.

Lines 100 to 220 initialise the matrices and set up the window for drawing.

Due to the design of QL graphics, line 100 can be freely omitted.

At the bottom of the listing are three PROCedures:

MATVEC multiplies a vector with a matrix (ie. the vector is transformed by this matrix) and MATVECS does the same for a list of vectors, just calling MATVEC for each individual vector. MATVEC(S) is written in a dimension independent way, just to show how that can be done; there is no check on the parameters, just to save space.

POLYDRAW draws a closed polygon from a supplied list of two-dimensional points.

```
100 WINDOW 448,200,32,16
110 SCALE 8,-5,-4: PAPER 0: CLS
120 :
```

(continues on next page)

(continued from previous page)

```
130 DIM ROT(2,2): rc=1/SQRT(2)
140 ROT(1,1)=rc: ROT(1,2)=rc
150 ROT(2,1)=-rc: ROT(2,2)=rc
160 DIM SQZ(2,2): SQZ(1,1)=.5: SQZ(2,2)=.5
170 :
180 DIM x(4,2), y(4,2), z(4,2)
190 x(1,1)=-1: x(1,2)= 1
200 x(2,1)= 1: x(2,2)= 1
210 x(3,1)= 1: x(3,2)=-1
220 x(4,1)=-1: x(4,2)=-1
230 :
240 INK 5: POLYDRAW x
250 MATVECS y,ROT,x
260 INK 3: POLYDRAW y
270 MATVECS z,SQZ,y
280 INK 7: POLYDRAW z
290 :
300 PAUSE: CLS
310 DIM ROTSQZ(2,2)
320 INK 5: POLYDRAW x
330 MATMULT ROTSQZ,ROT,SQZ
340 MATVECS z,ROTSQZ,x
350 INK 7: POLYDRAW z
360 :
370 :
380 DEFine PROCedure MATVECS (vectors2, matrix, vectors1)
390   LOCal i
400   FOR i=1 TO DIMN(vectors1)
410     MATVEC vectors2(i),matrix,vectors1(i)
420   END FOR i
430 END DEFine MATVECS
440
:
450 DEFine PROCedure MATVEC (vector2, matrix, vector1)
460   REMark vector2 = matrix * vector1
470   LOCal i,j
480   FOR i=1 TO DIMN(vector2)
490     vector2(i)=0
500     FOR j=1 TO DIMN(matrix,2)
510       vector2(i)=vector2(i)+matrix(i,j)*vector1(j)
520     END FOR j
530   END FOR i
540 END DEFine MATVEC
550 :
560 DEFine PROCedure POLYDRAW (vectors)
570   LOCal i
580   POINT vectors(1,1),vectors(1,2)
590   FOR i=2 TO DIMN(vectors), 1
600     LINE TO vectors(i,1),vectors(i,2)
```

(continues on next page)

(continued from previous page)

```
610   END FOR i
620 END DEFine POLYDRAW
```

**NOTE**

Normally the product of two matrices A*B is not the same as B*A, however, the matrices ROT and SQZ in the above example are an exception to this rule. Replace line 330 with: 330 MATMULT ROT-SQZ,SQZ,ROT and nothing will change.

**CROSS-REFERENCE**

See *MATINV* for another example of using *MATMULT*.

## 20.15 MATPLOT

| | |
|---|---|
| Syntax | MATPLOT array [{,\|;}] |
| Location | Math Package |

This command takes a two-dimensional array and draws the points set out by the array (the first dimension identifies the number of points and the second the co-ordinates) to the default window used by LINE (normally #1). The array must be declared in the following way (an array which does not fall into this category will cause an error):

```
DIM array (points,1)
```

points is the total number of points (less one) set out in the array, with array(p,0) the x-coordinate and array(p,1) the y- coordinate of point number p-1. If a comma (,) appears after the name of the array MATPLOT will connect each point with its successor by a line.

On the other hand, if a semicolon (;) appears after the name of the array, an additional line is drawn between the first point and the last point.

These lines are drawn using the QDOS line drawing routine and therefore suffer from the same problems as the LINE command. For those of you still uncertain of the possible uses of this command, a little hint: the addition of a semicolon to the the parameter will always enclose the set of lines which have been set out, thereby making this command ideal for creating all types of shapes (for example dodecahedrons)! MATPLOT supports INK, PAPER, OVER and FILL.

**Example**

The following fractal generator was written by John de Rivaz in SuperBASIC and optimised by Simon N. Goodwin. Originally, both the calculation and drawing was done in one loop which was a bit faster (10-20%) than the following version (this calculates all points in one loop and then uses MATPLOT to draw them quickly, creating a second internal loop). Another disadvantage compared to the original version is the increase in memory usage because all points have to be stored:

```
100 MODE 4: WINDOW 512,256,0,0: PAPER 0: CLS
110 SCALE 20,-14,-10: iterations=10000
120 DIM pts(iterations-1,1): x=0: y=0
130 FOR loop=0 TO iterations-1
140   pts(loop,0)=x: pts(loop,1)=y
150   sy=0: IF x<0 THEN sy=-1: ELSE IF x THEN sy=1
```

(continues on next page)

(continued from previous page)

```
160   xx=y-sy\*(ABS(x-.9))^.5: y=1.01-x: x=xx
170 END FOR loop
180 INK 7: MATPLOT pts
```

A nice modification of the above example would be to:

- Replace MODE 4 with MODE 8 in line 100;
- Delete line 180;
- Add the following block:

```
180 REPeat loop
190 FOR n=1 TO 7
200 INK n
210 MATPLOT pts
220 END FOR n
230 END REPeat loop
```

It's up to you to produce more variants!

**NOTE**

The output of MATPLOT cannot be redirected to any other window and specifically any program which uses MATPLOT (eg. the above example) cannot be compiled. So it is perhaps best to forget about MATPLOT.

**CROSS-REFERENCE**

*MATPLOT_R* draws the figure relative to the graphic cursor. *POINT* draws a single point to any screen, *BLOCK* can also be used to plot points, especially of variable size. *PLOT*, *APOINT* and POINT_ABS plot points in absolute co-ordinates, directly to screen memory, ignoring windows.

## 20.16 MATPLOT_R

| | |
|---|---|
| Syntax | MATPLOT_R array [{,\|;}] |
| Location | Math Package |

This command is the same as MATPLOT except that the output is drawn relative to the graphic cursor.

**CROSS-REFERENCE**

*POINT* and all other commands related to graphics move the graphic cursor.

## 20.17 MATPROD

| | |
|---|---|
| Syntax | MATPROD (array) |
| Location | Math Package |

The function MATPROD returns the product of the array's values, so array is not allowed to be a string array.

**Example**

Can you see why MATPROD and FACT return the same number for every n?

```
100 n=8: DIM a%(n)
110 MATSEQ a%
120 PRINT MATPROD(a%) ;" = ";
130 PRINT FACT(n+1)
```

**NOTE**

MATPROD is not compatible with Turbo and Supercharge.

**CROSS-REFERENCE**

*MATPROD* is almost identical to *MATSUM* except that it returns the product rather than the elements' sum; so have a look at *MATSUM* which is also more useful.

## 20.18 MATREAD

| | |
|---|---|
| Syntax | MATREAD array |
| Location | Math Package |

The command MATREAD initialises the array (of any type) by reading each element from DATA lines. Since MATREAD does the same as the following routine:

```
FOR i1=0 TO DIMN(array,1)
  FOR i2=0 TO DIMN(array,2)
    ...
    READ array(i1, i2)
    ...
  END FOR i2
END FOR i1
```

all of the normal errors of READ may occur.

**Example**

The following example is identical to MATSEQ a%

```
100 DIM a%(3,2)
110 MATREAD a%
120 :
130 DATA 1, 2, 3, 4
140 DATA 5, 6, 7, 8
150 DATA 9,10,11,12
```

is identical to MATSEQ a%.

**CROSS-REFERENCE**

*MATINPUT*

## 20.19 MATRND

| | |
|---|---|
| Syntax | MATRND array or<br>MATRND array% [[ ,minval%] ,maxval%] |
| Location | Math Package |

This command initialises all of the elements of an integer or floating point array with random numbers. Their default range depends on the type of array: for integer arrays, the values range from -32768 to 32767, whereas for floating point they range between 0 and 1.

MATRND selects the range itself if there is just one parameter, but for integer arrays only, an extended syntax allows you to specify another range (as in the second variant).

If just a maximum value maxval% is specified then values range from 0 to maxval%, if a minimum minval% is additionally given then values range from minval% to maxval%.

MATRND will reject any non-integer parameters for the second syntax.

**Examples**

```
DIM array%(4,3,2), array(1,2): min%=10
MATRND array
MATRND array%
MATRND array%,100
MATRND array%,min%,100
```

**NOTE**

Like all other MAT... commands, MATRND cannot be compiled with Supercharge or Turbo.

**WARNING**

MATRND allows a string array as a parameter. This leads to odd results and can possibly hang the machine.

**CROSS-REFERENCE**

The random values chosen by *MATRND* can be influenced by *RANDOMISE*.

## 20.20 MATSEQ

| | |
|---|---|
| Syntax | MATSEQ array |
| Location | Math Package |

The command MATSEQ initialises the array (which must be a numeric array) with a constantly increasing set of integer numbers: 1 2 3 4 5 6...

There is not really much use for MATSEQ except for demonstration.

Array can be either a floating point or integer variable. No strings are allowed.

**CROSS-REFERENCE**

*MATIDN* is a useful means of initialising an array, *MATEQU* can be used to set all elements of an array to a certain value. It is worth noting that any square matrix created with *MATSEQ* cannot be inverted with *MATINV* because the determinant *DET* of that matrix is always zero:

```
100 n=30: DIM m(n,n), minv(n,n)
110 MATSEQ m
120 MATINV minv,m
```

This always fails at line 120 because *DET*(m) = 0.

## 20.21 MATSUB

| | |
|---|---|
| Syntax | MATSUB difference,matrix1,matrix2 |
| Location | Math Package |

Provided that the parameters of the command MATSUB fulfil the same conditions as for MATADD, MATSUB will store the difference between matrix1 and matrix2 in difference. Difference(...) = matrix1(...) - matrix2(...). Two or all of the parameters can be identical, so:

```
MATSUB a,a,a
```

and:

```
MATSUB a,b,a
```

etc. are valid.

**CROSS-REFERENCE**

*MATADD*!

## 20.22 MATSUM

| | |
|---|---|
| Syntax | MATSUM (array[%]) |
| Location | Math Package |

This function calculates the sum of all of the elements of the supplied array. array can be any floating point or integer array, but not a string array. The latter leads to error -15 (bad parameter). Array can be any number of dimensions, although the following example uses just one dimension for demonstration reasons.

**Example**

If you stored a lot of values, eg. temperatures, in an array and want to find the average temperature, you have to divide the sum of the temperatures by the number of values. Obviously the operation of adding temperatures can take quite some time for a large data base, so this is a point where MATSUM helps:

```
100 values% = 200: DIM temp%(values%)
110 :
120 PRINT#0,"random initialisation..."
```

(continues on next page)

(continued from previous page)

```
130 MATRND temp%,-20,30
140 PRINT#0,"equalising";
150 FOR equalize = 1 TO 10
160   FOR i = 0 TO values%-1
170     temp%(i) = ( temp%(i) + temp%(i+1) ) / 2
180   END FOR i
190   PRINT#0,".";
200 END FOR equalize
210 :
220 PRINT#0\\"drawing..."
230 WINDOW 448,200,32,16: SCALE 100,0,0
240 PAPER 3: CLS: INK 7: OVER 0
250 dist = 160 / values%: yoff = 50
260 FOR i = 0 TO values%-1
270   x1 = i*dist: x2 = x1+dist
280   y1 = temp%(i) + yoff: y2 = temp%(i+1) + yoff
290   LINE x1,y1 TO x2,y2
300 END FOR i
310 :
320 PRINT#0,"find medium..."
330 tmed = MATSUM(temp%) / values%
340 INK 3: OVER -1
350 LINE 0,tmed+yoff TO x2,tmed+yoff
```

The important line is 330 where MATSUM is used. Lines 150 to 200 transform the random values to more realistic temperatures: you won't find any country where outside temperature jumps from -20 to +30 degrees Celsius in one day! The number of equalize loops can be freely chosen.

This is also true for values%, the figure adapts itself to the number of values (see dist in line 250).

**NOTE**

A program using MATSUM cannot be compiled with Turbo or Super-charge.

**CROSS-REFERENCE**

*MATRND* initialises an array with random values. *MATPROD* is very similar to *MATSUM* except that it finds the product of an array's elements. *MATMEAN* finds the mean value of a matrix's values directly, so line 330 could be replaced with:

```
330 tmed = MATMEAN(temp%)
```

## 20.23 MATTRN

| | |
|---|---|
| Syntax | MATTRN array1, array2 |
| Location | Math Package |

The command MATTRN takes numeric arrays of two dimensions or string arrays of three dimensions and reads each row of array2, placing it in the corresponding column of array1.

It is obligatory that both arrays have the same type and are exactly DIMed to the needs of MATTRN.

The first dimension of array1 must be equal to the second of array2 and the first dimension of array2 must be equal to the second of array1.

For strings, additionally, the third dimensions of both arrays have to be equal:

```
DIM array1(x,y), array2(y,x)
DIM array1%(x,y), array2%(y,x)
DIM array1$(x,y,z), array2$(y,x,z)
```

So array1 and array2 can only be of identical dimensions for square matrices. In all other cases the contents of array1 are not modified.

**Example**

```
100 DIM A%(2,3), B%(3,2)
110 MATRND B%,9: PRINT B%!\
120 MATTRN A%,B%: PRINT A%!\
130 MATTRN B%,A%: PRINT B%!\
```

## 20.24 MAX

| | |
|---|---|
| Syntax | MAX ($x^1$ *[,$x^i$]*) |
| Location | Math Package, MINMAX2 |

This function must be given at least one number as a parameter - it will then return the highest value out of the given list of parameters.

**Example**

```
PRINT MAX ( 2, 5, -10, 3.2 )
```

will print 5.

**CROSS-REFERENCE**

*MIN*. See also *MAXIMUM* and *MATMAX*.

## 20.25 MAX_CON

| | |
|---|---|
| Syntax | error = MAX_CON(#channel%, x%, y%, xo%, yo%) |
| Location | DJToolkit 1.16 |

If the given channel is a 'CON_' channel, this function will return a zero in the variable 'error'. The integer variables, 'x%', 'y%', 'xo%' and 'yo%' will be altered by the function, to return the maximum size that the channel can be *WINDOW*'d to.

'x%' will be set to the maximum width, 'y%' to the maximum depth, 'xo%' and 'yo%' to the minimum x co-ordinate and y co-ordinate respectively.

For the technically minded reader, this function uses the IOP_FLIM routine in the pointer Environment code, if present. If it is not present, you should get the -15 error code returned. (BAD PARAMETER).

**EXAMPLE**

```
7080 DEFine PROCedure SCREEN_SIZES
7090   LOCal w%,h%,x%,y%,fer
7100   REMark how to work out maximum size of windows using iop.flim
7110   REMark using MAX_CON on primary channel returns screen size
7120   REMark secondaries return maximum sizes within outline where
7130   REMark pointer environment is used.
7140   w% = 512 : REMark width of standard QL screen
7150   h% = 256 : REMark height of standard QL screen
7160   x% = 0
7170   y% = 0
7180   :
7190   fer = MAX_CON(#0,w%,h%,x%,y%) : REMark primary for basic
7200   IF fer < 0 : PRINT #0,'Error ';fer : RETurn
7210   PRINT'#0 : ';w%;',';h%;',';x%;',';y%
7220   :
7230   fer = MAX_CON(#1,w%,h%,x%,y%) : REMark primary for basic
7240   IF fer < 0 : PRINT #0,'Error ';fer : RETurn
7250   PRINT'#1 : ';w%;',';h%;',';x%;',';y%
7260   :
7270   fer = MAX_CON(#2,w%,h%,x%,y%) : REMark primary for basic
7280   IF fer < 0 : PRINT #0,'Error ';fer : RETurn
7290   PRINT'#2 : ';w%;',';h%;',';x%;',';y%
7300 END DEFine SCREEN_SIZES
```

## 20.26 MAX_DEVS

| | |
|---|---|
| Syntax | how_many = MAX_DEVS |
| Location | DJToolkit 1.16 |

This function returns the number of installed directory device drivers in your QL. It can be used to *DIM*ension a string array to hold the device names as follows:

```
1000 REMark Count directory devices
1010 :
1020 how_many = MAX_DEVS
1030 :
1040 REMark Set up array
1050 :
1060 DIM device$(how_many, 10)
1070 :
1080 REMark Now get device names
1090 addr = 0
1100 FOR devs = 1 to how_many
1110   device$(devs) = DEV_NAME(addr)
1120   IF addr = 0 THEN EXIT devs: END IF
1130 END FOR devs
```

**CROSS-REFERENCE**

*DEV_NAME*.

## 20.27 MAXIMUM

| | |
|---|---|
| Syntax | MAXIMUM [ ( array ) ] or<br>MAXIMUM ( *[ value ]* ) |
| Location | Minmax (DIY Toolkit - Vol Z) |

The effect of this function depends on the parameter supplied. It is however an extremely fast way of comparing values. If no parameter is supplied, then the greatest possible floating point number supported by the QL is returned - this is equivalent to 1.61585 e616.

If a single parameter is supplied which is a single dimensional floating point array, then MAXIMUM will return the value of the largest number stored within that array.

If you want to compare the values of an integer array, then use MAXIMUM% (a 'bad parameter' is generated with this (MAXIMUM) function).

If, however, you use the second variant to pass a list of values (either numbers or variables), then the highest value out of those parameters will be returned. Please note that you cannot pass an array in this instance - it is therefore the same as MAX.

**Example**

```
PRINT MAXIMUM
```

Returns 1.61585e616

```
DIM x(3): x(0)=10: x(1)=200: x(2)=2.5: x(3)=50.4
PRINT MAXIMUM (x)
```

Returns 200.

```
PRINT MAXIMUM (100, ax ,21*10+ac)
```

Returns the highest value.

**NOTE**

This function cannot be compiled with Supercharge or Turbo if you intend to pass an array as the parameter.

**CROSS-REFERENCE**

*MATMAX*, *MAXIMUM%* and *MAX* are similar. Refer also to *MINIMUM* and *MINIMUM%*.

## 20.28 MAXIMUM%

| | |
|---|---|
| Syntax | MAXIMUM% [ ( array% ) ] or<br>MAXIMUM% ( *[ value ]* ) |
| Location | Minmax (DIY Toolkit - Vol Z) |

This function is exactly the same as MAXIMUM except that it only accepts integer parameters and is therefore able to work much more quickly. As with MAXIMUM, you can use this function to find the highest value in an array, provided that the first variant is used, and the array is a single dimensional integer array. If no parameter is supplied, then the greatest possible integer number supported by the QL is returned - this is equivalent to 32767.

**Example**

```
PRINT MAXIMUM%
```

Returns 32767

```
DIM x%(3): x%(0)=10: x%(1)=200: x%(2)=2: x%(3)=50
PRINT MAXIMUM% (x%)
```

Returns 200.

```
PRINT MAXIMUM% (100, ax ,21*10+ac)
```

Returns the highest value as an integer.

**NOTE**

This function cannot be compiled with Supercharge or Turbo if you intend to pass an array as the parameter.

**CROSS-REFERENCE**

*MATMAX*, *MAXIMUM* and *MAX* are similar. Refer also to *MINIMUM%*.

## 20.29 MB

| | |
|---|---|
| Syntax | MB |
| Location | Minerva |

Early versions of Minerva (pre v1.97) did not have built-in MultiBASICs and they had to be EXECuted from disk. However, you could make them resident by linking in the file Mulib_rext with the LRESPR command and then this command, MB would be available to start up MultiBASIC interpreters. This is not a very convenient way of starting MultiBASICs as you cannot pass parameters to the MultiBASIC, nor can you use the command to run filter programs.

**NOTE**

This command is redundant on Minerva v1.97+, whereby MultiBASICs can be started up using EXEC pipep.

**CROSS-REFERENCE**

See *SBASIC* and *EW*. Also see *QUIT*. Check out the appendix on Multiple BASICs.

## 20.30 MD

| | |
|---|---|
| Syntax | MD subdir |
| Location | Beuletools (Needs Level-2 Drivers) |

This command is just used as an abbreviation for the MAKE_DIR command on Level-2 (and higher) floppy/ winchester/ ramdisk drivers.

**CROSS-REFERENCE**

An alternative would be to rename *MAKE_DIR* with *NEW_NAME*. See also *MAKE_DIR*.

## 20.31 MERGE

| | |
|---|---|
| Syntax | MERGE device_filename or<br>MERGE [device_]filename (Toolkit II) |
| Location | QL ROM, Toolkit II |

This command is similar to LOAD *except* that it does not clear the current program and variables out of memory prior to loading the given program file. Neither is the screen cleared, which enables loading pictures to be shown on screen whilst the main program loads.

This means that any line numbers which appear in the program currently in memory and which are repeated in the program file will be *overwritten* by the lines in the program file, whereas any new lines will be inserted into the program in memory.

Again, any lines without line numbers are automatically executed as they are loaded into memory. This could therefore be used within a program to execute a 'command file' stored on a directory device (however, see below).

**Example**

A short program - when typed in, save this using the command:

```
SAVE mdv1_test1_bas
```

```
10 REMark Test1
20 PRINT 'The Sinclair QL'
```

Now, type NEW and enter the following short program:

```
5 REMark Test
20 PRINT 'An old program line'
30 PRINT 'SuperComputer'
```

Now, enter the command:

```
MERGE mdv1_test1_bas
```

followed by:

```
LIST
```

and the following will now form the program in memory:

```
5 REMark Test
10 REMark Test1
20 PRINT 'The Sinclair QL'
30 PRINT 'SuperComputer'
```

**NOTE 1**

Unfortunately, if you MERGE a file of direct commands (ie. a program file without line numbers), only the first line will be read and the file will be left open, making it impossible to change the disk/microdrive cartridge. Some compilers provide commands to ensure that the file is closed and all of the commands executed.

Minerva and Toolkit II close the file, but still only the first command is executed, unless the MERGE command is used from within a program (in which case, the whole of the command file is executed). SMS ensures that MERGE works in both of these circumstances.

**NOTE 2**

When writing command files, ensure that the lines are all checked thoroughly before saving them without the line numbers, since a 'bad line' error on such a file may crash the QL. However, if Toolkit II is present, this makes a safe recovery, reporting 'bad line'.

**NOTE 3**

MERGE can become confused if used from within a PROCedure or FuNction. Minerva and Toolkit II both report 'Not Implemented'.

**NOTE 4**

On Minerva v1.86, MERGE could become a little confused when used within a program.

**NOTE 5**

When used within a program MERGE and MRUN are the same.

**NOTE 6**

Since Toolkit II v2.22 (and on the Minerva version), MERGE has refused to try and load a file which does not have a file type of 0 - see FTYP.

**SMS NOTE**

MERGE follows the same rules for finding a program name as the LOAD command.

**CROSS-REFERENCE**

*MRUN* is very similar. See *LOAD* and *SAVE*. *DO* is also very similar to *MERGE*.

## 20.32 MIDINET

| | |
|---|---|
| Syntax | MIDINET |
| Location | SMSQ/E, ATARI Emulators |

A file MIDINET_rext is provided with SMSQ/E and the Emulators for the Atari computers which allows you to set up a Network using the MIDI ports provided on the Atari computers.

Once the Network has been set up with the necessary leads, and MIDINET_rext been loaded on all computers in the Network, the command MIDINET should be issued to start up the fileserver Job on each computer. This creates a background Job called 'MIDINET' which is similar to the 'Server' Job created by FSERVE.

The two fileservers are very similar in operation in that they both allow other computers to access the resources of the Master machine over the Network. However, MIDINET has built-in protection for files which can prevent other users in a Network accessing sensitive files. This is implemented by means of recognising files which *start with* a specific series of characters:

| Characters | Effect |
|---|---|
| *H or *h | These files cannot be accessed over the Network. Any attempt to use these files by a Slave Machine will return 'Not Found' errors. |
| *R | These files are Read Only over the Network. |
| *D | These files cannot be accessed over the Network and will return 'Not Implemented' - this prevents direct sector access. |

**CROSS-REFERENCE**

*MNET* is needed to control the Network. See also *FSERVE* and *SERNET*. See the Appendix on Networks for further details.

## 20.33 MIN

| | |
|---|---|
| Syntax | MIN ($x^1$ *[,$x^i$]*) |
| Location | Math Package, MINMAX2 |

This function must be given at least one number as a parameter - it will then return the lowest value out of the given list of parameters.

**Example**

```
100 INPUT "a ="!a
110 INPUT "b ="!b
120 FOR x=MIN(a,b) TO MAX(a,b): PRINT x
```

**CROSS-REFERENCE**

*MAX* is *MIN*'s counterpart. Compare *MINIMUM* and *MATMIN*.

## 20.34 MINIMUM

| | |
|---|---|
| Syntax | MINIMUM [ ( array ) ] or<br>MINIMUM ( *[ value ]* ) |
| Location | Minmax (DIY Toolkit - Vol Z) |

The effect of this function depends on the parameter supplied. It is however an extremely fast way of comparing values.

If no parameter is supplied, then the smallest possible floating point number supported by the QL is returned - this is equivalent to -1e614.

If a single parameter is supplied which is a single dimensional floating point array, then MINIMUM will return the value of the smallest number stored within that array. If you want to compare the values of an integer array, then use MINIMUM% (a 'bad parameter' is generated with this function if you attempt to use it for integers).

If, however, you use the second variant to pass a list of values (either numbers or variables), then the smallest value out of those parameters will be returned.

Please note that you cannot pass an array in this instance - it is therefore the same as MIN.

**Example**

```
DIM x(3): x(0)=10: x(1)=200: x(2)=2.5: x(3)=50.4
PRINT MINIMUM (x)
```

Returns 2.5

**NOTE**

This function cannot be compiled with Supercharge or Turbo if you intend to pass an array as the parameter.

**CROSS-REFERENCE**

*MATMIN*, *MINIMUM%* and *MIN* are similar. Refer also to *MAXIMUM* and *MAXIMUM%*.

## 20.35 MINIMUM%

| | |
|---|---|
| Syntax | MINIMUM% [ ( array% ) ] or<br>MINIMUM% ( *[ value ]* ) |
| Location | Minmax (DIY Toolkit - Vol Z) |

This function is exactly the same as MINIMUM except that it only accepts integer parameters and is therefore able to work much more quickly. As with MINIMUM, you can use this function to find the smallest value in an array, provided that the first variant is used, and the array is a single dimensional integer array. If no parameter is supplied, then the smallest possible integer number supported by the QL is returned - this is equivalent to -32768.

**NOTE**

This function cannot be compiled with Supercharge or Turbo if you intend to pass an array as the parameter.

**CROSS-REFERENCE**

*MATMIN*, *MINIMUM* and *MIN* are similar. Refer also to *MAXIMUM%*.

## 20.36 MISTake

| | |
|---|---|
| Syntax | MISTake |
| Location | QL ROM |

MISTake is a keyword which will only rarely ever be found. It cannot be inserted into a program from the keyboard. Instead, it is generated internally whenever LOAD, LRUN, MERGE or MRUN commands are used and a line in the file being loaded cannot be parsed (ie. if it would generate a 'bad line' error if typed in at the keyboard).

Rather than reporting an error and stopping the loading process, the word MISTake is inserted in the offending line after the line number. If you then try to RUN the offending line, a 'Bad Line' error will be generated (under SMS the error 'MISTake in program' is reported.

You can however EDIT the offending line - you must delete the word MISTake as well as correcting the error before the line will be accepted by the parser. Once this is done, then the program should run as normal.

**NOTE**

Unfortunately, QREF (from Liberation Software) cannot find lines containing MISTake - in order to do this, you need a much more complex system such as MasterBasic+ (from Ergon Development).

**CROSS-REFERENCE**

Please see *LOAD* and *MERGE* about loading a SuperBASIC program in general.

## 20.37 MKF$

| | |
|---|---|
| Syntax | MKF$ (float) |
| Location | BTool |

This function returns a string containing the internal representation of a floating point number (which is stored as six bytes).

**CROSS-REFERENCE**

*CVF*, *MKI$*, *MKS$*, *MKL$*, *PEEK_F*, *POKE_F*

## 20.38 MKI$

| | |
|---|---|
| Syntax | MKI$ (integer%) where integer% = -32768..32767 |
| Location | BTool |

The function MKI$ returns a string containing the internal representation of an integer number (which is stored as two bytes).

**Example**

```
MKI$(11111)
```

Would return the string "+g", because:

```
CODE("+")*256 + CODE("g")
```

Equals 11111.

**CROSS-REFERENCE**

*CVI%* is the opposite function. *MKF$*, *MKL$*, *MKS$*

## 20.39 MKL$

| | |
|---|---|
| Syntax | MKL$ (longint) where longint = -2*INTMAX-1..2*INTMAX+1 |
| Location | BTool |

This function returns a string containing the internal format of a long integer number (which is stored as four bytes).

**CROSS-REFERENCE**

*CVL* is the complementary function. *MKI$*, *MKF$*, *MKS$*

## 20.40 MKS$

| | |
|---|---|
| Syntax | MKS$ (string$) |
| Location | BTool |

This function returns a string containing the internal format of a string {which is stored as two bytes indicating the length of the string (as returned by MKI$) and the string itself}.

**Example**

```
MKS$("Test") = CHR$(0)&CHR$(4) & "Test"
```

because:

```
MKI$ (4)
```

returns the string CHR$(0)&CHR$(4).

**CROSS-REFERENCE**

*CVS$*, *MKI$*, *MKF$*, *MKL$*

## 20.41 MNET

| | |
|---|---|
| Syntax | MNET station |
| Location | SMSQ/E, ATARI Emulators |

This command is similar to the NET command in that it sets the Network Station number of the machine on which it is issued. The only difference is that here it sets the station number for the MIDINET Network (as opposed to QNET).

**CROSS-REFERENCE**

See *MNET%*, *MNET_USE* and *NET*. Also please see *MIDINET*, *SERNET* and *FSERVE*.

## 20.42 MNET%

| | |
|---|---|
| Syntax | MNET% |
| Location | SMSQ/E, ATARI Emulators |

This function returns the current station number of the computer as set with MNET.

**CROSS-REFERENCE**

See *MNET*. *NET_ID* is similar.

## 20.43 MNET_OFF

| | |
|---|---|
| Syntax | MNET_OFF |
| Location | SMSQ/E, ATARI Emulators |

This command turns the MIDINET driver off temporarily so that you can use the MIDI ports independently.

**CROSS-REFERENCE**

See *MNET_ON*.

## 20.44 MNET_ON

| | |
|---|---|
| Syntax | MNET_ON |
| Location | SMSQ/E, ATARI Emulators |

This command switches the MIDINET driver back on after it has been disabled with MNET_OFF.

**CROSS-REFERENCE**

See *MNET_OFF*. Also see *MIDINET*.

## 20.45 MNET_S%

| | |
|---|---|
| Syntax | MNET_S% (station) |
| Location | SMSQ/E, ATARI Emulators |

This function enables you to check whether a machine with the specified station number is connected to the MIDINET. This can be useful to prevent the problem of the Network retrying several times before failing when asked to send or read data from a Network station which does not exist.

**CROSS-REFERENCE**

See *MNET*.

## 20.46 MNET_USE

| | |
|---|---|
| Syntax | MNET_USE id |
| Location | SMSQ/E, ATARI Emulators |

Due to the fact that MIDINET Networks can be run on computers alongside SERNET Networks and even QNET Networks, it may be necessary to alter the identification letter used to access facilities on other computers in the Network. The default letter id is n (as with FSERVE), but this can be set to any other single letter by using this command. However, you should avoid letters which already appear as the first letter in another device driver (see DEVLIST).

**Example**

```
MNET_USE m
OPEN #3,m2_con_512x256a0x0
```

Open an input channel covering the screen on station number 2 in the MIDINET Network.

**CROSS-REFERENCE**

See *MNET* and *MIDINET*. Refer also to*MNET_S%*.

## 20.47 MOD

| | |
|---|---|
| Syntax | x MOD y |
| Location | QL ROM |

This operator returns the value of x to modulus y. This is defined as x-(x DIV y)*y. If x or y is not an integer value, then it is rounded to the nearest integer (compare INT). On non-SMS implementations the answer and both parameters must lie within the range -32768…32767. On SMS, the answer and both parameters can lie anywhere within roughly - 2e9…2e9.

**Examples**

```
PRINT 13 DIV 5
```

gives the result 3. This is because 13 DIV 5 is 2, 2 multiplied by 5 is 10, 13 minus 10 is 3.

```
PRINT 13.4 MOD 1.5
```

gives the result 1 (13 MOD 2).

**NOTE 1**

MOD has problems with the value -32768: PRINT -32768 MOD -1 gives the result -1 on most implementations. On Minerva v1.76 (or later) and SMS v2.77+ it gives the correct result, being 0.

**NOTE 2**

If you write a program for SMSQ/E which uses values outside the range -32768…32767, this will not work on non-SMSQ/E machines - instead of:

```
PRINT x MOD y
```

you will need to use:

```
PRINT x - (INT(x / y) * y)
```

**CROSS-REFERENCE**

*DIV* returns the integer part of x divided by y. Also, please see the alternative version of *MOD*.

## 20.48 MOD

| | |
|---|---|
| Syntax | MOD (x,y) |
| Location | Math Package |

The function MOD returns the value x-(DIV(x,y)*y), ie. the value of x to modulus y, in a similar fashion to the ROM based operator MOD.

However, this version is not limited to a range of -32768 to 32767, but will accept parameters in the range -INTMAX to INTMAX. Because both versions of MOD return the integer remainder of a division, x MOD 0 or MOD(x,0) lead to an overflow error, because division by zero is undefined.

**NOTE 1**

Both versions of MOD can be used in the same program, although the Turbo and Supercharge compilers will not accept this alternative form.

**NOTE 2**

If you try to use a program compiled under Turbo or Supercharge after loading the Math Package, if the program uses the normal SuperBASIC operator MOD or DIV, an error will be generated and the program will refuse to work!

**CROSS-REFERENCE**

*DIV MOD* (ROM version)

## 20.49 MODE

| | |
|---|---|
| Syntax | MODE mode% or<br>MODE screen_mode [,display_type] (Minerva, Q-Emulator, Amiga-QDOS v3.23+) or<br>MODE [screen_mode [,display_type]](PEX only) |
| Location | QL ROM, PEX |

The original QDOS operating system will only recognise two display modes: Low resolution and High resolution. However, the following MODEs are currently set aside for use by QDOS compatible systems:

| MODE | Resolution | Colours | System |
|---|---|---|---|
| 2 | 640 x 400 | 2 | SMS-2 |
| 4 | <=1000 x 400 | 4 | SMS-2 |
| 4 | 768 x 280 | 4 | ST/QL, Ext. MODE 4 |
| 4 | <=1024 x 1024 | 4 | QVME |
| 4 | <=800 x 600 | 4 | QXL, QXL II, QPC |
| 4 | 512 x 256 | 4 | QDOS and others |
| 8 | 256 x 256 | 8 | QDOS and others |
| 8 | 256 X 256 | 4 | ST/QLs |
| 12 | 256 x 256 | 16 | THOR XVI |

The MODE command is used to select the mode and redraw all windows. Without Qjump's Window Manager WMAN, the screen mode is set globally, whereas if WMAN (or SMSQ/E) is installed (this is highly recommended), MODE will only affect the current job.

The parameter mode% can be any legal integer between -32768 and 32767. However, to ensure compatibility with other systems one of the above four values should be used. Normally if a system does not support the mode type selected, MODE 4 is selected.

The MODE command also resets the current status of UNDER, FLASH, CSIZE and OVER.

Without specialised software, only one screen mode can be used at a time (even with the specialised software contained in the Quanta library, the screen can only be split in two horizontally).

The second variant is supported on Minerva, Q-Emulator (for the MacIntosh), PEX and the Amiga QDOS Emulator (v3.23+) and allows you to dictate the type of display used. The display_type can be one of four values (the default is -1):

| Display_type | Effect |
|---|---|
| 0 | Set to monitor mode |
| 1 | Set to TV (625 lines) mode (European) |
| 2 | Set to TV (525 lines) mode (American) |
| -1 | Leave display type as it is |

On the PEX variant, if you do not specify any parameters, MODE will default to MODE 4,0

**NOTE 1**

Normally, High resolution is described as MODE 4 because this value represents a characteristic of the mode (4 colours) as well as setting it. Equally, MODE 8 stands for Low resolution. However, with the ability of QDOS to access much higher resolution screens, these terms now tend to be somewhat unnecessary.

**NOTE 2**

Unfortunately for Minerva users who wish to run software in dual screen mode, current versions of the Pointer Interface do not allow you to have different MODEs on each of the two screens (the pointer interface fails to recognise that a program is running on the second screen only and does not therefore affect the main display screen located at $20000). Speedscreen may also give problems in Minerva's dual screen mode unless the 'p' version is used.

**NOTE 3**

If you want to make your programs appear more professional, you should always seek to cut out unnecessary MODE commands (see RMODE), also because of the fact that MODE tends to re-draw all of the current windows (clearing them in their current paper and border colours as it works), it is always an idea to ensure that all currently opened windows are set to black paper and black (or no) border before issuing this command.

**NOTE 4**

On an American JSU QL (which was adapted for use with the American 525-line TV picture, as opposed to the British 625-line TV picture), only 192 lines of pixels are allowed instead of the normal 256 in MODE 4 and MODE 8 (when the QL is linked to a TV). There are less and less users using their QL with a TV set nowadays and therefore this can be largely ignored. In any event, software should generally still run on an American QL without modification (the lower number of available lines on the TV screen ensures that pictures still appear to retain the same height/width ratio).

**NOTE 5**

If you are planning to use the dual screen mode, it is essential that you ensure that the current screen is also the displayed screen before opening windows or using the MODE command - see below.

**NOTE 6**

The standard screen modes are MODE 4 and MODE 8. MODE 8 is however only supported on a limited number of implementations. It is supported by the original QL, some early ST-QL Emulators and Amiga-QDOS (v3.23+).

**DUAL SCREEN MODE**

Minerva and some other implementations allow you to have two screens which can both be accessed by the user (and can be switched between by pressing <CTRL><TAB>). Each of these two screens (if you are in dual screen mode), can support a different mode. In order to cater for these new features, screen_mode is very complex, and to make it worse, it is important to know which screen is the default screen (see DEFAULT_SCR).

Programs which use the normal MODE commands will still work under dual screen mode, since the new version of the MODE command will only work if the display_type is specified.

When the QL is first started, unless you choose <F3> or <F4> (on Minerva), only one screen will be available for use by programs, otherwise Minerva is placed into Dual Screen Mode.

In the dual screen mode, after starting up the QL, the default screen is scr0 (located at $20000 - the normal QL display screen). The second screen (scr1) is located at $28000 and is known as the Other Screen.

To make matters worse, each job present in the QL's memory will be allocated its own default screen, being the current default when it was started. A job can therefore alter its own current default screen without upsetting the rest of the system.

Before proceeding any further it is important to realise that the Displayed Screen (what you can see on your TV/monitor) and the Default Screen are not necessarily one and the same thing. Oh, it is also important to know that a screen can also be either visible or blank (this is so that work can be prepared on a screen without the user being able to see the process). Perhaps some definitions might help:

**Displayed Screen:**
This is the screen which is currently in front of the user on his/her monitor or TV.

**Default Screen:**
The screen on which a program's windows will be opened and upon which the normal MODE 4 and MODE 8 commands will have an effect.

**Other Screen:**
The opposite to the Default Screen (ie. if the Default Screen is scr0, then the Other Screen will be scr1).

**Visible Screen:**
This means that the specified screen can actually be seen by the user.

**Blank Screen:**
The specified screen is invisible to the user (allows background work to be carried out).

That's the definitions out of the way, and hopefully, they will provide a better understanding of what is to follow. The command:

```
OPEN#3,scr_448x200a32x16
```

will open a new window on the current Default Screen. After this, any subsequent commands using #3 will be shown on that screen (whether or not it is still the current Default Screen).

Problems may exist with some Toolkit functions which do not check to see where the screen starts for the given window, and just assume that the screen starts at $20000.

Unfortunately, current versions of Lightning and the Pointer Interface introduce various problems to the Dual Screen Mode, the most important one of which is that the screen will not be re-drawn unless the current screen is also the Displayed Screen.

Another plus to the altered MODE command is that there is no forced re-draw of all the current windows unless you specify that this must be carried out (or if you use the original MODE variants).

In order to try and explain the new display_mode parameters, it is easier to split it into two sections: toggling current values and setting absolute values.

**Toggling the Screen Parameters**

This uses the format MODE 64+n,-1, where:

| n | Effect | From: | To: |
|---|---|---|---|
| 1 | Toggle Other Screen | Visible | Blank |
| 2 | Toggle Default Screen | Visible | Blank |
| 4 | Toggle Other Screen Mode | 4-colour | 8-colour |
| 8 | Toggle Default Screen Mode | 4-colour | 8-colour |
| 16 | Toggle Displayed Screen | scr0 | scr1 |
| 32 | Toggle Default Screen | scr0 | scr1 |

Adding together different values of n will combine these effects (although if one of the values is to be 32, the default screen will be toggled before anything else is carried out).

**Examples**

```
MODE 64+16,-1: PAUSE: MODE 64+16,-1
```

Show both screens.

```
MODE 64+4+8,-1
```

Toggle the mode of both screens Details of the values used to set absolute screen parameters appear on the next page.

**Setting Absolute Screen Parameters**

This uses the format MODE -128 + m - 256 * t + c, -1

where:

- m = k1*n1 + k2*n2 + k3*n3 +...
- t = n1 + n2 + n3 +...n
- c = (see below)
- n can have the same values as above, depending on which effect is to be altered.
- k1, k2, k3 etc. have the following effect upon the corresponding values of n1, n2, n3, etc.

| k | Sets n to: |
|---|---|
| 0 | The 'from..' column above |
| 1 | The 'to..' column above |

| c | Effect |
|---|---|
| 0 | Do not redraw any screens |
| -16384 | Re-draw the Other Screen |
| 32768 | Re-draw the Default Screen |
| 16384 | Re-draw both screens |

Again, different effects can now be combined with relevant values for each n and k. If you wish to toggle any effects at the same time, simply add the corresponding value of n to the first parameter. Some Minerva manuals do not have the correct formula for calculating these values, which can lead to some peculiar results. Changing the default screen will again take precedence to all other changes.

**Dual Screen Examples**

```
MODE 4
```

change the Default Screen to MODE 4 and re-draw all currently opened windows on the Default Screen.

```
MODE 64+32,-1
```

toggle current Default Screen.

```
MODE 64+32+16,-1
```

toggle current Default Screen and show to user.

```
MODE -17791,-1
```

blank out Other Screen and then force it into 4-colour mode, redrawing all windows, Where does -17791 come from? The formula given above:

```
-128 + m - 256*t + c
```

Into which we substitute the following:

```
t = 1 + 4
m = 1*1 + 0*4
c = -16384
```

Care must however be taken when opening channels if the two screens are in different modes - on versions of Minerva earlier than v1.97, if you open a channel on the non-Displayed Screen, it will have the characteristics of a window opened in the mode of the Displayed Screen (although sadly this does not mean that you can have a MODE 4 window in the middle of a MODE 8 screen). To ensure that the current Default Screen is the current Displayed Screen, use:

```
MODE -128 + DEFAULT_SCR * 16 - 256 * 16, -1
```

**Q-EMULATOR NOTE**

Q-Emulator for the Apple MacIntosh computer supports Minerva's dual screen mode and the extended MODE command.

**AMIGA-QDOS NOTE**

From v3.23, the Amiga-QDOS Emulator also supports Minerva's dual screen mode and the extended MODE command. Before this version, it did not support MODE 8. Even now, FLASH is not supported in MODE 8.

**WARNING 1**

Changing the display_type may have odd effects, especially if Qjump's Button Frame (part of QPAC2) is present.

**WARNING 2**

On pre JS ROMs, if you open a screen (scr_) or console (con_) channel after a MODE command, the ink and paper colours for the new channel could both be 0 (black).

**WARNING 3**

On pre Minerva ROMs, MODE alters the value contained in the system variable SYS.DTYP (also known as SV.TMOD) which normally contains a value from 0…2 showing the type of TV/Monitor the QL is set up for. Speedscreen, the Pointer Environment and Lightning all fix this.

**CROSS-REFERENCE**

*RMODE* can be used to read the current screen mode (and even whether the second screen is available) and *DEFAULT_SCR* will tell you which is the current default screen. *SCREEN*(#3) will tell you the address of the start of the screen on which window #3 is situated. *DISP_SIZE* can be used to set the size of the displayed screen on extended resolutions.

## 20.50 MONTH%

| | |
|---|---|
| Syntax | MONTH% [datestamp] |
| Location | SMSQ/E |

This function complements the *DATE* and *DATE$* functions, by returning the month number corresponding to the given datestamp, or current date, if no datestamp was given.

**Examples**

```
PRINT MONTH% (0)
```

will print the month part of the QL's epoch, 1 for January

```
PRINT MONTH%
```

will print the current month number, (1…12, starting with January).

**CROSS-REFERENCE**

See *DATE*, *YEAR%*.

## 20.51 MORE

| | |
|---|---|
| Syntax | MORE [#ch,] filename |
| Location | MORE (DIY Toolkit - Vol V) |

This command adds a quite sophisticated file viewing facility to the QL which far surpasses the simple Toolkit II VIEW command.

In its simplest form, MORE will open a channel to the specified filename (adding the data default directory if the file does not exist) and display it in the specified window channel (default #1). If #ch does not refer to a window or is #0, then bad parameter will be reported. The file will then be displayed in the specified channel, one window full at a time. #0 is used by the command to display the length of the file in bytes and the number of the last byte displayed in the window.

You can move around the file by using the following keys:

- <ENTER> - Allows you to enter a file position to look at (this will be the first byte displayed in the window).
- <ALT><UP> - This moves back up the file one page at a time.
- <ALT><DOWN> - This moves down the file one page at a time.
- <DOWN> - Move down the file one line.
- <ESC> - Leave MORE.

MORE can however, also be used to look at the QL's memory (or that on a networked computer) by using the MEM device. In this mode, only the address of the last byte on screen is shown in #0 - there is no file length. For example:

```
MORE #2,MEM
```

will allow you to use MORE to page through the whole of the QL's memory.

```
MORE #2,n2_MEM
```

allows you to page through the whole of another computer's memory.

```
OPEN #3,MEM7_60p: PRINT #3,'Hello World': CLOSE #3
```

creates a permanent buffer (MEM7) and stores two words in it. If you follow the above by:

```
MORE #2,MEM7
```

then you will be able to look at the contents of the buffer MEM7.

**NOTE**

Trying to use MORE on anything other than files or MEM devices (for example on named pipes) will cause problems - press <CTRL><SPACE> a few times to escape from this.

**CROSS-REFERENCE**

Refer to the Devices Appendix for more details on MEM.

Compare:

```
COPY flp1_test_bas to SCR
```

and:

```
VIEW flp1_test_bas
```

## 20.52 MOUSE_SPEED

| | |
|---|---|
| Syntax | MOUSE_SPEED [#ch,] acceleration, wakeup |
| Location | SMSQ/E for QPC |

This function adjusts the mouse acceleration and wake up factor. The acceleration factor is of no consequence to QPC2. The wakeup values, however, may still be set. They range from 1 to 9, with 1 being the most sensitive.

## 20.53 MOUSE_STUFF

| | |
|---|---|
| Syntax | MOUSE_STUFF [#ch,] hot$ |
| Location | SMSQ/E for QPC |

This function adjusts the string that is stuffed into the keyboard queue when the middle mouse button is pressed (or both left and right buttons are pressed simultaneously). The string cannot be longer than two characters, but this is enough to trigger any hotkey, which in turn, can do almost anything.

**Example**

```
MOUSE_STUFF '.'
```

Generates a dot if middle mouse button is pressed.

```
MOUSE_STUFF CHR$(255) & '.'
```

Generates hotkey <Alt><.> which will activate whatever has been defined on that key combination.

## 20.54 MOVE

| | |
|---|---|
| Syntax | MOVE [#ch,] distance |
| Location | QL ROM |

The QL supports a simplified means of drawing pictures known as turtle graphics. This was based upon an early educational tool, whereby simple commands could be entered into a computer to drive a small robot turtle which moved around the floor and held a pen. This pen could either be up in which case the turtle would just move around, or down in which case a line would be left by the turtle on the floor as it moved.

When a window is first opened, an invisible turtle appears at the graphics origin (altered with SCALE) facing to the right, with its pen in the up position.

The command MOVE forces the turtle in the specified window (default #1) to move in the current direction by the specified distance.

The actual distance moved on screen depends on the current SCALE applicable to that window. If distance is negative, the turtle will move backwards. MOVE always works from the current graphics cursor position, and after using this command, the current graphics cursor is placed at the turtle's position on screen. MOVE is affected by the current INK colour, FILL and also OVER, just like any other graphics command.

**Example**

A simple procedure to draw a shape of a set number of equal length sides:

```
100 DEFine PROCedure POLYGON (chan, sides, side_length)
110   TURNTO #chan,0: PENDOWN #chan
120   FOR k = 1 TO sides
130     MOVE #chan, side_length
140     TURN #chan, 360 / sides
150   END FOR k
155   PENUP #chan
160 END DEFine
```

Try for example, POLYGON #2,5,10.

**NOTE**

The THOR XVI v6.40 tended to crash when using turtle graphics, especially if a channel number was specified.

**CROSS-REFERENCE**

*PENDOWN* forces the pen into the down position, leaving a trail on screen. *PENUP* allows the turtle to move without leaving a trail. *TURN* and *TURNTO* allow you to alter the direction of the turtle.

## 20.55 MOVE_MEM

| | |
|---|---|
| Syntax | MOVE_MEM destination, length |
| Location | DJToolkit 1.16 |

This procedure will copy the appropriate number of bytes from the given source address to the destination address. If there is an overlap in the addresses, then the procedure will notice and take the appropriate action to avoid corrupting the data being moved. Most moves will take place from source to destination, but in the event of an overlap, the move will be from (source + length -1) to (destination + length -1).

This procedure tries to do the moving as fast as possible and checks the addresses passed as parameters to see how it will do this as follows :-

- If both addresses are odd, move one byte, increase the source & destination addresses by 1 and drop in to treat them as if both are even, which they now are!
- If both addresses are even, calculate the number of long word moves (4 bytes at a time) that are to be done and do them. Now calculate how many single bytes need to be moved (zero to 3 only) and do them.
- If one address is odd and the other is even the move can only be done one byte at a time, this is quite a lot slower than if long words can be moved.

The calculations to determine which form of move to be done adds a certain overhead to the function and this can be the slowest part of a memory move that is quite small.

**EXAMPLE**

```
MOVE_MEM SCREEN_BASE(#0), SaveScreen_Addr, 32 \* 1024
```

## 20.56 MOVE_POSITION

| | |
|---|---|
| Syntax | MOVE_POSITION #channel, relative_position |
| Location | DJToolkit 1.16 |

This is a similar procedure to *ABS_POSITION*, but the file pointer is set to a position relative to the current one. The direction given can be positive to move forward in the file, or negative to move backwards. The channel must of course be opened to a file on a directory device. If the position given would take you back to before the start of the file, the position is left at the start, position 0. If the move would take you past the end of file, the file is left at end of file.

After a MOVE_POSITION command, the next access to the given channel, whether read or write, will take place from the new position.

**EXAMPLE**

```
MOVE_POSITION #3, 0
```

moves the current file pointer on channel 3 to the start of the file.

```
MOVE_POSITION #3, 6e6
```

moves the current file pointer on channel 3 to the end of the file.

**CROSS-REFERENCE**

*ABS_POSITION*.

## 20.57 MRUN

| | |
|---|---|
| Syntax | MRUN device_filename or<br>MRUN [device_]filename (Toolkit II) |
| Location | QL ROM, Toolkit II |

This command is similar to MERGE except that once the two programs have been merged, if MRUN was issued as a direct command, then the merged program is RUN from line 1. However, if MRUN was used from within the program, the statement following the MRUN statement is executed, thus making this command the same as MERGE when used within a program.

**CROSS-REFERENCE**

See *MERGE*!

## 20.58 MSEARCH

| | |
|---|---|
| Syntax | MSEARCH (add1 TO add2, tofind$) |
| Location | MSEARCH (DIY Toolkit - Vol X) |

This function is very similar to the Tiny Toolkit version of the SEARCH function except that it performs an extremely fast case-independent search through memory (much more quickly than other implementations).

**CROSS-REFERENCE**

See search and *TTFINDM* also. *SEARCH_MEM* is a variant on this version.

## 20.59 MT

| | |
|---|---|
| Syntax | MT (i,n) |
| Location | Toolfin |

The function MT returns the value of $(1+i)^n$ where i and n can be any floating point numbers. Instead of reporting an overflow error for values which cannot be computed (eg. i=-1, n=-1) MT returns 1. If the returned value would be too large, a modulated value is returned. It is therefore imperative that the programmer takes care that the parameters are correct, otherwise the return values may not make much sense.

**Example 1**

MT gives you a factor which indicates the increase (i>1) or decrease (i<1) of capital at an interest rate i over a number of periods n. The gain is known as compound interest. If you give any sum to a bank at an interest rate of five percent (ie. annual 5 per 100 increase) for (say) ten years, you will gain 62.9% because: MT(0.05, 10) = 1.628895

**Example 2**

```
MT(1/n,n)
```

approximates EXP(1) for large values of n.

**CROSS-REFERENCE**

*VA*, *VFR*, *VAR*, *TCA*, *TNC*, *TEE*, *RAE*, *RAFE*

## 20.60 MTRAP

| | |
|---|---|
| Syntax | MTRAP key [,d1 [,d2 [,d3 [,a0 [,a1 ]]]]] or<br>MTRAP key\jobnr [,d2 [,d3 [,a0 [,a1 ]]]]] |
| Location | TRAPS (DIY Toolkit Vol T) |

This command is similar to QTRAP in that it allows you to access the machine code TRAP #1 system calls directly. Unless you are using the second variant, you will need to pass at least one parameter, the operation key to be carried out (this is equivalent to the value in D0 when TRAP #1 is performed). The other parameters allow you to pass the various register values which may be required by the system calls.

The second variant is useful for when you are using a TRAP #1 call which requires a job ID - you can merely pass the jobnr of the required job, obtained from the JOBS list (rather than having to set D1 to the Job ID). For example to force remove Job 12, use the command:

```
MTRAP 5\12,0,0
```

**WARNING**

Several TRAP #1 calls can crash the computer - make certain that you know what you are doing!

**CROSS-REFERENCE**

See *IO_TRAP*, *QTRAP* and *BTRAP*. *REMOVE_TASK* and *RJOB* are better for removing Jobs. Any return parameters can be read with *DATAREG* and *ADDREG*. Refer to the QDOS/SMS Reference Manual (Section 15) for details of the various system TRAP #3 calls.

CHAPTER

# TWENTYONE

# KEYWORDS N

## 21.1 NDIM

| | |
|---|---|
| Syntax | NDIM (array) |
| Location | Math Package |

Identical to *NDIM%*.

## 21.2 NDIM%

| | |
|---|---|
| Syntax | NDIM% (array) |
| Location | NDIM |

This function returns the number of dimensions of a given array. DIMmed variables of any type (floating point, integer and string) are legal parameters.

**Example**

```
DIM test% (10,50,2)
PRINT NDIM% (test%)
```

gives the answer 3.

**NOTE**

String arrays often have one more dimension than the number of elements which they can hold. This extra dimension sets the maximum length of each element, for example:

```
DIM name$(100,20)
```

sets aside space in the array name$ for 100 strings, each of which can be a maximum of 20 characters long:

```
PRINT NDIM%(name$) returns 2.
```

**CROSS-REFERENCE**

*DIMN* finds the defined size of each dimension and can be used to replace *NDIM%*. *DIM* declares an array. *NDIM* is exactly the same as *NDIM%*.

## 21.3 NET

| | |
|---|---|
| Syntax | NET x% |
| Location | QL ROM |

This command sets the computer's station number for use in a network to x%. A station can have any integer number in the range 1..128 (although see Note below). Each station in the network should have a different station number to avoid confusion. When the computer is first switched on (or reset), it is given the station number 1.

**Example**

```
NET 12
```

sets the station number to 12.

**NOTE**

Many implementations allow a station number in the range 1..127 (SMS allows 1..255), although there is still a maximum of 64 computers which can be linked into the network at any one time using standard QLs.

Auroras allow more to be linked together.

**CROSS-REFERENCE**

Please see the Appendix concerning Networks. See also *SNET* and *MNET*.

## 21.4 NETBEEP

| | |
|---|---|
| Syntax | NETBEEP delay, pulses |
| Location | FLEXYNET (DIY Toolkit - Vol Y) |

This command allows you to send signals through the QL's network ports - if you plug a pair of earphones into the network port (instead of a Network lead), you will be able to hear the sound generated - this can even work alongside the QL's BEEP command to provide the QL with rudimentary polyphonic sound. In fact, some of the Spectrum emulators for the QL use this feature to provide the QL with Spectrum-like sound.

The first parameter tells the command the length of the delay between pulses sent to the Network port - the higher the delay, the lower will be the pitch.

The second parameter tells the command the number of pulses to send to the port - each pulse will send an electrical signal through the network port (equivalent to turning a switch on and then back off).

You can also use this command (in conjunction with NETPOLL on other computers in the Network to test the speed settings for Flexynet) - simply send a series of bytes over the Flexynet (using NETBEEP) and check that they have been received correctly at the other end by using NETPOLL. If the bytes are incorrect, you may need to increase the delay.

**NOTE**

The units used by both parameters are quite arbitrary and therefore some experimentation may be required.

**CROSS-REFERENCE**

Refer to the Appendix on Networks to find out more about Flexynet. See *NETPOLL* which allows you to read signals sent through the network ports. *NETRATE* allows you to alter the speed of the Flexynet and *NETREAD* / *NETSEND* can be used to read and send multiple bytes.

## 21.5 NETPOLL

| | |
|---|---|
| Syntax | NETPOLL address, bytes |
| Location | FLEXYNET (DIY Toolkit - Vol Y) |

This command allows you to sample electrical signals sent over the QL's Network, which can be used to decode any digital data stream, such as information sent by cassettes tapes (this method has been used by some Spectrum emulators for the QL to allow you to load Spectrum games direct from tape).

You need to pass two parameters - the address of a place in memory where the data which is read is to be stored, followed by the number of bytes which can be stored at the address.

For each byte to be stored at the specified address, NETPOLL 'listens' to the QL's Network ports and whenever an electrical pulse is received by the port (for example as sent by NETBEEP), then NETPOLL counts the length of this pulse and sets the data byte to the relevant value, moving onto the next available byte (or returning to BASIC if it has reached the end of the storage area).

The value of the data byte will be between 0 and 254, with the delay being the difference between the value and 255 (hence a value of 127 shows a delay of approximately twice a value of 254). If a value is 255, this indicates that the timer ran out before a pulse arrived.

**NOTE**

An odd number of bytes at the storage area will be rounded up, so that an even number of pulses will always be read.

**CROSS-REFERENCE**

The amount of time that *NETPOLL* will wait for a pulse is affected by *NETRATE*0,0,x. See *NETBEEP* which sends signal tones along the network ports. Also see *NETRATE*,*NETREAD* and*NETSEND*.

## 21.6 NETRATE

| | |
|---|---|
| Syntax | NETRATE transmit_delay, reception_delay, timeout |
| Location | FLEXYNET (DIY Toolkit - Vol Y) |

This command is at the heart of the Flexynet philosophy and allows you to alter the speed of the QL's networks (as regards NETSEND and NETREAD), so that you can match the speed of the network ports to the various computers which are linked together over the network. This enables different machines to talk to each other substantially more quickly than using the Network drivers supplied with the QL or Toolkit II.

Values are sent over the Network ports as pulses equivalent to bits, with eight bits representing one byte (0…255) - the pulse is an electrical signal, either 1 or 0 (on or off). The three parameters are all in arbitrary units and if they are specified as zero, any existing value will remain unchanged. They are used as follows:

- Transmit_delay - this specifies the amount of time that the sending machine will alter the voltage for on the network to signify either a 1 or a 0 bit. This needs to be higher than the reception delay on the receiving machine as the receiving process is fundamentally slower than the transmission process. The higher the value, the longer the delay.
- Reception_delay - this specifies the amount of time Flexynet will wait for changes in the voltage over the network ports.

  If the change occurs after Flexynet has counted up to the reception_delay value, then a bit of 0 is assumed, otherwise a bit of 1. Once eight bits have been received then a byte made up of those eight bits is stored in memory. For example: CHR$(10) is represented by the bits 0 0 0 0 1 0 1 0 This can be seen by:

```
PRINT BIN$ (10,8)
```

- Timeout - this represents a timing loop which Flexynet will wait for the next pulse over the net - if nothing is received in this time, then a 'not complete' error will be reported. This value will need to exceed the reception_delay parameter by a comfortable margin. The easiest way to match up the required parameter values for two machines connected using Flexynet is to send a copy of one machine's screen to the other machine and compare the display. To do this, enter the command:

```
NETREAD 131072,32768
```

  on the receiving machine, then enter the command:

```
NETSEND 131072,32768
```

  on the sending machine.

  If the NETRATE parameters are incorrect, you will notice that the displays do not match - either increase the reception_delay on the receiving machine or increase the transmit_delay on the sending machine, making notes of the values which you have tried at either end.

It is difficult to give any advice on the parameters to use as it depends on the expansion boards being used with your particular QL, as well as the speed of the ZX-8301 chip which forms part of the QL's motherboard. However, the author cites the following test results:

- Standard QL to Standard QL (both with code in ROM or fast RAM such as CST 512K expansion board):

```
NETRATE 5,3,127
```

  on both machines

- Gold Card on British QL to Gold Card on Foreign QL:

```
NETRATE 8,4,0
```

on both machines.

- Gold Card on Foreign QL to Gold Card on British QL:

```
NETRATE 33,12,0
```

  on both machines

- Standard QL (with code in ROM or fast RAM such as CST 512K expansion board) to Gold Card on Foreign QL:

```
NETRATE 2,2,127
```

on the Standard QL:

```
NETRATE 20,7,127
```

on the Gold Card QL

**NOTE**

Flexynet will not work on machines which do not use a 68000 or 68008 chip (for example QXLs or Super Gold Card), unless the Cache is disabled (see CACHE_OFF). It also requires an expanded machine to work properly.

**CROSS-REFERENCE**

Refer to the Networks Appendix for further details. See *NETVAR%* which allows you to read the various speed settings. Also see *NETREAD* and *NETSEND*

## 21.7 NETREAD

| | |
|---|---|
| Syntax | NETREAD address, bytes |
| Location | FLEXYNET (DIY Toolkit - Vol Y) |

This command will attempt to read the specified number of bytes over the Network port using the Flexynet driver and store any bytes it receives at the area in memory starting with the specified address. This area of memory should therefore really be set aside with ALCHP or RESPR before use (unless you know that the area of memory can be altered (such as the screen memory - see SCREEN).

**NOTE 1**

This command should only be used in conjunction with NETSEND. Do not attempt to use any other network drivers whilst one machine has used this command.

**NOTE 2**

This command must be used before the NETSEND command is issued, if it is to catch the data sent by the transmitting machine.

**CROSS-REFERENCE**

See *NETRATE* which allows you to set the speed of the receiving machine to match the speed of the sending machine. Also see *NETSEND*.

## 21.8 NETSEND

| | |
|---|---|
| Syntax | NETSEND address, bytes |
| Location | FLEXYNET (DIY Toolkit - Vol Y) |

This command will attempt to send the specified number of bytes over the Network port using the Flexynet driver, reading the bytes to be sent from the area in memory starting with the specified address.

**NOTE**

Please refer to the notes given for NETREAD.

**CROSS-REFERENCE**

See *NETRATE* which allows you to set the speed of the transmitting machine to match the speed of the receiving machine. Also see *NETREAD*.

## 21.9 NETVAR%

| | |
|---|---|
| Syntax | NETVAR% (parameter) |
| Location | FLEXYNET (DIY Toolkit - Vol Y) |

This function returns the various values set with the NETRATE command. The possible values of parameter are:

| Parameter | Meaning |
|---|---|
| 1 | Return the Transmission Delay |
| 2 | Return the Reception Delay |
| 3 | Return the Timeout |

**CROSS-REFERENCE**

See *NETRATE*.

## 21.10 NET_ID

| | |
|---|---|
| Syntax | NET_ID |
| Location | THOR XVI |

This function returns the computer's station number set with *NET*. Other ROMs can also find out their station number by using:

```
PRINT PEEK (SYS_VARS+55)
```

or:

```
PRINT PEEK (!!55)
```

(the latter syntax being accepted by Minerva and SMS).

**CROSS-REFERENCE**

See *NET* which sets the station number. Also see *MNET%* and *SNET%*

## 21.11 NEW

| | |
|---|---|
| Syntax | NEW |
| Location | QL ROM, Toolkit II |

If the command NEW is issued under the interpreter, the current SuperBASIC program is removed from memory, the values of all variables are forgotten, all channels owned by the interpreter (job 0) which have a number equal to or higher than #3 are closed and the windows #0, #1, #2 are cleared (in this order).

The Minerva, THOR XVI and Toolkit II versions of NEW also disable WHEN ERRor clauses. A bug in JS and MGx ROMs meant that these clauses could not be disabled once activated.

From within a compiled program, NEW removes the job from which it was issued (ie. the current job).

**WARNING**

All data stored in variables is lost.

**CROSS-REFERENCE**

*CLEAR*, *KILL_A*. Inside compiled programs, *NEW* and *STOP* are effectively the same. *RESET* clears the whole system by restarting it.

## 21.12 NEWCHAN%

| | |
|---|---|
| Syntax | NEWCHAN% |
| Location | Function (DIY Toolkit - Vol R) |

When writing / designing SuperBASIC programs, it is essential that you try to keep the channel numbers as small as possible for two reasons - compilers only allow a fixed number of channels to be OPENed by a program (normally 16) and if you OPEN #100,scr (for example), space has to be created by SuperBASIC in the channel table for channels #1 to #99, thus wasting a lot of memory if those channels are not used.

This function can therefore be quite useful - it looks at the channel table and returns the number of the next available channel number which can be OPENed.

**Example**

After:

```
NEW
PRINT NEWCHAN%
```

will always return 3, as the only channels OPEN will be #0, #1 and #2.

**CROSS-REFERENCE**

*OPEN* allows you to open a channel. *FOPEN* and similar functions will automatically open the next available channel number.

## 21.13 NEW_NAME

| | |
|---|---|
| Syntax | NEW_NAME old_name$, new_name$ |
| Location | TinyToolkit |

This command allows all keywords, variables, procedures, functions and device names to be renamed, whether they are in RAM or ROM (except device names, which must be RAM based), BASIC or machine code implementations. If a program is loaded when the command is issued, then all references in that program to the given name will also altered.

**Examples**

- FORMAT can destroy a lot of data. To avoid a catastrophe when an alien, unknown BASIC program formats your hard disk for example, you could rename FORMAT:

```
NEW_NAME "FORMAT" TO "FORMAT_MEDIUM"
```

  Note that you would need to issue this command before loading the program!

- Creating algorithms is very easy and fast in SuperBASIC, especially if short variable names like i ,n, q1 are used. But even the author him/herself may have difficulty in understanding source code full of such meaningless names. The obvious solution: Rename them! - for example:

```
NEW_NAME "d","dog"
```

  BASIC programs loaded in memory are amended completely and permanently - at once.

- If you prefer to see all names in capital letters, run this short program:

```
100 adr=BASICP(32)
110 REPeat all_names
120   length=PEEK(adr)
130   IF NOT length THEN EXIT all_names
140   name$=PEEK$(adr+1,length)
150   NEW_NAME name$,UPPER$(name$)
160   adr=adr+length+1
170 END REPeat all_names
```

**NOTE**

A name may be up to 255 characters long, and because it is only stored once (in the name table) and represented in a tokenised program by symbols pointing to the name table, the actual speed of operation will not be slowed down by using longer names.

**WARNING**

It is possible to rename FORMAT to FORMAT! (for example) but FORMAT! is an illegal name, can no longer be called from BASIC and may crash some advanced implementations of SuperBASIC.

**CROSS-REFERENCE**

*ZAP* and *KEY_RMV* remove a resident keyword. See *REPLACE* and *ALIAS* as well.

## 21.14 NEXT

| | |
|---|---|
| Syntax | NEXT loop_variable (inside FOR loops) or<br>NEXT loop_name (inside REPeat loops) or<br>NEXT(SMS only) |
| Location | QL ROM |

This command forces the program to make the next pass in a loop structure - the next command to be processed is the first after the relevant FOR or REPeat instruction. NEXT can be used in both loop structures, FOR and REPeat.

**NOTE**

If a FOR loop has already reached its last value, NEXT will have no effect.

**SMS NOTE**

The loop_name / loop_variable do not need to be specified, in which case NEXT merely makes the program make the next pass of the latest defined FOR or REPeat loop. If NEXT does not appear within a loop structure, the error 'unable to find an open loop' will be reported. If however, NEXT is followed by a loop_name or loop_variable and that does not correspond to a currently open loop, the error 'undefined loop control variable' will be reported.

**CROSS-REFERENCE**

You must study *FOR* or *REPeat* before using *NEXT*. *EXIT* leaves a loop.

## 21.15 NFS_USE

| | |
|---|---|
| Syntax | NFS_USE newdrive, drive1 [,drive2 [..., drive8]] or<br>NFS_USE [newdrive] |
| Location | Toolkit II, THOR XVI |

Two QLs, both fitted with Toolkit II on EPROM (or SMS) and connected via a network cable, can use Toolkit II's file server which is activated by the FSERVE command. All of the devices on the other QL (provided the Server job is running on that QL) can then be accessed as if they were a normal device on the QL wishing to use the facilities. This is achieved by prefixing the device name by: n<netnr>_, eg:

```
DIR n2_flp1_
```

will show the directory of flp1_ on station number 2.

*NET* sets this station number.

Two problems do however arise from using this technique: Firstly, it is a bit annoying to have to type n2_flp1_. Secondly, a lot of programs check the validity of a device by checking if the length is five characters, the first three characters of which must be letters, and the fourth character of which must be a digit from 1 to 8 with an underscore at the end. These programs therefore only allow device names such as ram6_, mdv1_, etc. To fool these programs (and also to shorten names):

```
NFS_USE
```

can be used to create a new device which has a shorter name. The first parameter is the name of the new drive which can be any description up to four characters long (there is no need to include a number or underscore). After this up to eight parameters (each of which can be up to a maximum of 15 characters) can follow which specify the drive which should be accessed as (for example): flop1_, … flop8_. It is neither possible to rename a local drive with:

```
NFS_USE test,ram1_
```

(error -12), nor indirectly with NET1:

```
NFS_USE test,n1_ram1_
```

The second example can be entered but any attempted access to test1_ will lead to a Network aborted message after half a minute of complete silence.

The second syntax is used to remove a specified set of definitions (or, if no parameter is supplied, then all definitions will be removed) which have been created with NFS_USE.

**Examples**

```
NFS_USE flop,n2_flp1_,n3_flp1_
```

creates a device name flop where flop1_ refers to flp1_ on QL2 and flop2_ to flp1_ on QL3. NFS_USE flop clears the above definition.

NFS_USE without any parameters clears all such definitions.

**NOTE**

Devices can be shared by several remote QLs. Although a file can be read by several jobs (or QLs) at the same time, QDOS will ensure that a file cannot be opened by one job (or QL) for writing to whilst another is trying to read from it (or vice versa). If this occurs, then an error -9 (IN USE) will be reported.

**CROSS-REFERENCE**

QRD renames any local device. See also *FLP_USE*, *RAM_USE* and *DEV_USE*. *MIDINET* and *SERNET* set up similar fileservers to *FSERVE* - *NFS_USE* can be used with these fileservers also, provided that you use *SNET_USE* n or *MNET_USE* n to ensure that they are identified by the letter n.

## 21.16 NIX

| | |
|---|---|
| Syntax | NIX |
| Location | Beuletools |

Nix is a word from colloquial German and means nothing, which is nearly what this command does. The command is intended to help multitask Qmon, a monitor program published by Qjump. When the command is issued, a dummy job named Qmons Nix-Job is created. If the monitor is started to examine this job, for example by entering:

```
QMON con_,4
```

(assuming that the dummy job has the job number 4), Qmon can easily be switched on and off.

## 21.17 NO_CLOCK

| | |
|---|---|
| Syntax | NO_CLOCK |
| Location | THOR XVI |

The command NO_CLOCK removes the THOR's clock task which is invoked with CLOCK. This is a much cleaner method of removing the CLOCK job and can avoid some problems (see the note on CLOCK).

**CROSS-REFERENCE**

*CLOCK* starts up the clock job on the THOR's screen.

## 21.18 NOCAPS

| | |
|---|---|
| Syntax | NOCAPS |
| Location | BeuleTools |

If capslock was on, this command forces it to be switched off.

**CROSS-REFERENCE**

See *CAPS* for an example.

## 21.19 NOKEY

| | |
|---|---|
| Syntax | NOKEY |
| Location | KEYMAN |

This command temporarily disables all definitions of KEY. Pressing the key combination <CTRL><CAPS> toggles between enabled and disabled state, so this is partially equivalent to NOKEY.

**CROSS-REFERENCE**

See *KEY* for details.

## 21.20 NORM

| | |
|---|---|
| Syntax | NORM |
| Location | Beuletools |

This function returns the control codes needed to reset an EPSON compatible printer:

```
PRINT NORM
```

is the same as:

```
PRINT CHR$(27)&"@"
```

**Example**

```
LPRINT NORM
```

**CROSS-REFERENCE**

*BLD*, *EL*, *DBL*, *ENL*, *PRO*, *SI*, *NRM*, *UNL*, *ALT*,*ESC*,*FF*,*LMAR*, *RMAR*,*PAGDIS*, *PAGLEN*.

## 21.21 NOR_MSG

| | |
|---|---|
| Syntax | NOR_MSG |
| Location | ST/QL |

The file NOR_TRA_rext is supplied with the ST/QL Emulator which contains translation tables to allow the Emulator to use Norwegian. Once this file has been LRESPR'd, this function can be used to find the start of the message translation table to be used with the TRA command. You can use: TRA NOR_TRA,NOR_MSG to set up the printer and message translation tables for Norway.

**CROSS-REFERENCE**

See *GER_MSG* and *NOR_TRA*. Also see *TRA*.

## 21.22 NOR_TRA

| | |
|---|---|
| Syntax | NOR_TRA |
| Location | ST/QL |

This is the complementary function to NOR_MSG and points to the printer translation table for Norway contained in the file NOR_TRA_rext.

**CROSS-REFERENCE**

See *NOR_MSG*.

## 21.23 NOT

| | |
|---|---|
| Syntax | NOT x |
| Location | QL ROM |

NOT is an operator which does not combine two operands (unlike +, DIV or || for example) but only operates on one. In fact, it can be regarded as a function which returns a value depending on the operand, except that brackets are not needed around the operand.

NOT is a logical operator and returns either 1 if the operand is zero or 0 in any other case. The following function would work the same way:

```
100 DEFine FuNction NOT1 (x)
110   IF x=0 THEN RETurn 1: ELSE RETurn 0
120 END DEFine NOT1
```

or even shorter:

```
100 DEFine FuNction NOT2 (x)
110   RETurn x=0
120 END DEFine NOT2
```

**Example**

The above replacements of NOT demonstrate that it is not necessary at all to use NOT. But in context, NOT can clarify an expression and make program listings more readable. If is_lamp is a logical variable used to say whether something is a lamp (is_lamp=1) or not (is_lamp=0), there are (at least) two variants to write the status of is_lamp to the screen. Which is easier to read?

```
PRINT "This is ";: IF is_lamp=0 THEN PRINT "not ";PRINT "a lamp."
```

or:

```
PRINT "This is ";: IF NOT is_lamp THEN PRINT "not ";PRINT "a lamp."
```

Let's assume lamps is a variable counting lamps and you want to write out a message if there are no lamps left:

```
IF lamps=0 THEN PRINT "Sorry, we are out of lamps."
```

or:

```
IF NOT lamps THEN PRINT "Sorry, we are out of lamps."
```

Here, the first formulation, which does not use NOT is clearer.

Until now, the examples have shown that NOT can be used to improve the style of a program, but there are also ways to put NOT to practical use, especially if a logical variable is to be set depending on another logical variable.

For instance, this procedure will accept such a value as a parameter and convert it to its logical counterpart for its own use:

```
100 DEFine PROCedure MY_CIRCLE (x,y,r, filled)
110   IF filled THEN FILL 1
120   CIRCLE x,y,r
130   IF filled THEN FILL 0
140 END DEFine MY_CIRCLE
```

As IFs are relatively slow and FILL takes a logical parameter, the following variant is faster:

```
100 DEFine PROCedure MY_CIRCLE (x,y,r, filled)
110   FILL filled
120   CIRCLE x,y,r
130   FILL 0
140 END DEFine MY_CIRCLE
```

As FILL cannot handle parameters other than 0 and 1, if filled could have any value at all (not just 0 or 1), it would be necessary to change filled so that it was either 0 or 1, by an additional line:

```
105 IF filled THEN filled=1
```

NOT is ideal (although here a bit complex!) to avoid the IF and calculate filled directly:

```
110 FILL NOT(NOT filled)
```

**NOTE**

When dealing with logical variables, the use of NOT to toggle the value, for example:

```
filled = NOT filled
```

is invariably quicker than the use of an IF statement:

```
IF filled THEN filled = 0: ELSE filled = 1
```

**CROSS-REFERENCE**

Comparisons between any two values (or even two variables) is regarded as a numeric expression by SuperBASIC. *IF* handles actual numeric values.

## 21.24 NRM

| | |
|---|---|
| Syntax | NRM |
| Location | Beuletools |

This function returns the control codes to switch back to the normal font (Pica) on an EPSON compatible printer:

```
PRINT NRM
```

is the same as:

```
PRINT CHR$(27)&"P".
```

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*,*DBL*,*ENL*,*PRO*,*SI*,*UNL*,*ALT*,*ESC*,*FF*,*LMAR*,*RMAR*,*PAGDIS*, *PAGLEN*. *UPUT* allows you to send untranslated bytes to the printer.

## 21.25 NXJOB

| | |
|---|---|
| Syntax | NXJOB (job_ID, topjob_ID) or<br>NXJOB (jobname, topjob_ID) or<br>NXJOB (jobnr, jobtag, topjob_ID) |
| Location | Toolkit II |

This function will work downwards through a 'job tree' to find all of the current jobs which are both used by the given 'top job' and those which are used by that second set of jobs. A job tree may look something like this:

```
          SuperBASIC
               |
  +------------+-----------+
  |            |           |
EDITOR      QPAC-FILES    CLOCK
               |
      +--------+----------+
      |                   |
    QUILL               ABACUS
```

A job can be referred to either by its name (eg. 'Quill'), its job number and job tag (eg. 1,2) (shown by JOBS), or its job ID (a number calculated by job_number+65536*job_tag). These are always interchangeable, so assuming there is a job 'Test' with job number 1 and job tag 12.

```
PRINT NXJOB ('Test',0)
PRINT NXJOB (1,12,0)
PRINT NXJOB (65548,0)
```

are all the same.

Note that the top job ID must not be the job's name or job number and tag. You could, for instance, use:

```
PRINT NXJOB(0,0)
```

to find that SuperBASIC is using the Job QPAC-FILES. You must now follow that branch to its tip by using:

```
PRINT NXJOB ('QPAC-FILES',0)
```

to find the job ID of Quill.

```
PRINT NXJOB('Quill',0)
```

will then find the job ID of Abacus. Since Abacus is at the end of a main branch,

```
PRINT NXJOB('Abacus',0)
```

will find Clock.

Should you wish to merely find out which Jobs are used by QPAC-FILES, you can do this by altering the topjob_ID to the job_ID given for QPAC-FILES - eg:

```
PRINT NXJOB('QPAC-FILES',65535)
```

**Example**

A short program to work out the whole job tree belonging to SuperBASIC. This is very similar to the JOBS command, but displays the information slightly differently:

```
100 MODE 4
110 a=0: b=0
120 REPeat loop
130   c=NXJOB(a,b)
140   IF c=0: PRINT\'End of Job Table': STOP
150   a=c: IF LEN(JOB$(c))=0:PRINT'ANONYMOUS';: ELSE PRINT JOB$(c);
160   PRINT TO 15;'Priority = ';PJOB(c);
170   IF OJOB(c)=0: own$='SuperBASIC': ELSE own$=JOB$(OJOB(c))
180   PRINT TO 30;'Owner = ';own$
190 END REPeat loop
```

**CROSS-REFERENCE**

*PJOB*, *JOB$*, and *OJOB* also deal with the job tree. For jobs in general, see *JOBS*, *RJOB*, *SPJOB*, *SJOB*, *AJOB*.

CHAPTER

TWENTYTWO

# KEYWORDS O

## 22.1 ODD

| | |
|---|---|
| Syntax | ODD (number) |
| Location | TinyToolkit, BTool |

This function checks if the integer part of number can be divided by two without remainder, ie. whether it is an odd number or not. If the number is odd, the function will return 1, otherwise it will return 0. The ODD function could easily be duplicated in BASIC by the following function:

```
10 DEFine FuNction ODD (number%)
20   RETurn number% MOD 2
30 END DEFine ODD
```

## 22.2 OFF

| | |
|---|---|
| Syntax | OFF |
| Location | BTool |

This is a constant which returns 0. OFF and *ON* are intended to make listings more readable.

**Example**

```
FREEZE OFF
```

**CROSS-REFERENCE**

*ON*, *FALSE%*, *TRUE%*, *SET*

## 22.3 OJOB

| | |
|---|---|
| Syntax | OJOB (job_ID) or<br>OJOB (jobnr, tag) or<br>OJOB (jobname) |
| Location | Toolkit II |

This function will return the job_id of the 'owner' of the given job. Normally the owner of a job is the job which initiated it. So, if job 1 creates job 2 then job 1 is the owner of job 2. However, because jobs can execute other jobs without becoming their owner, generally the owner of a job is the job that will kill that job when it itself is removed. A negative job_ID points to the job which calls OJOB.

**CROSS-REFERENCE**

*JOB$*, *NXJOB* and *PJOB* return other information about a job. *JOBS* lists all jobs.

## 22.4 ON

| | |
|---|---|
| Syntax | ON |
| Location | BTool |

This is a constant which returns 1.

**Example**

```
FREEZE ON
```

**CROSS-REFERENCE**

*OFF*, *FALSE%*, *TRUE%*, *SET*

## 22.5 ON...GO TO

See *ON...GO SUB*.

## 22.6 ON...GO SUB

| | |
|---|---|
| Syntax | ON condition GO TO line$^1$ *[,line$^i$]* or<br>ON condition GO SUB line$^1$ *[,line$^i$]* |
| Location | QL ROM |

The QL supports a structure which enables the program to jump to specific lines depending upon the value of a variable. condition must be an integer expression which returns a value of one or more. After the command GO TO or GO SUB must appear a list of line numbers to jump to depending on the value of the condition. The value returned is then used to determine which of these line numbers will be jumped to, by counting the different options. If the result of the condition is zero, or more than the number of options, then the error 'Out of Range' will be returned. If ON...GO SUB is used, then when a RETurn is executed from within the sub-routine, the program will continue from the statement following ON..GO SUB.

**Example**

A procedure to print out the locations in an adventure might look like this:

```
100 no_of_locations=3
110 start=0
120 PRINT_LOC 2
```

(continues on next page)

(continued from previous page)

```
130 DEFine PROCedure PRINT_LOC (xa)
140   IF xa=0 OR xa>no_of_locations THEN PRINT 'Undefined Location':RETurn
150   ON xa+start GO SUB 170,180,190
160   PRINT 'What now?':RETurn
170   PRINT 'This is location 1':RETurn
180   PRINT 'This is location 2':RETurn
190   PRINT 'This is location 3':RETurn
200 END DEFine
```

For a simpler (and clearer) way of writing this PROCedure, please refer to the example given for SELect ON.

**NOTE**

ON can also be used with the SELect ON structure - please refer to *SELect ON* for further details.

**CROSS-REFERENCE**

These two commands can generally be replaced either by a calculated *GO SUB* or *GO TO* statement, or the *SELect ON* structure.

## 22.7 OPEN

| | |
|---|---|
| Syntax | OPEN #channel, device channel=0..32767 or<br>OPEN #channel, device, type (Minerva v1.80+ only) |
| Location | QL ROM, Toolkit II |

This is the general command used to open a channel to a device for input and/or output, so that data can be read from and written to the specified device. The channel number can be any integer greater than or equal to zero and should be kept as small as possible because QDOS allocates roughly 40 bytes for each possible channel number below the highest one. So if you open channel #1000, 40K of memory would be lost - only badly written programs need a thousand channels.

After the channel has been OPENed, if a program needs to access that device in the future, it can do so by passing that channel number to the relevant keyword.

Actually, a dozen channels should be sufficient and the Turbo compiler strictly limits the highest channel number to 15, while QLiberator allows you to configure this to the user's needs via a $$chan directive. The compilers allocate memory for all of the channels when a job is created so that the channel table of the job is independent of other jobs and cannot be extended or decreased.

Under the interpreter, the channel table can be freely extended but not decreased - only NEW and KILL_A clear the channel tables.

When talking about devices, it is necessary to note the difference between drives (file drivers) and serial devices:

- A drive is a medium where files can be stored (eg. floppy disks or microdrive cartridges). Since there can always be several drives of a given type, drive names contain a drive number from 1 to 8. Data is always stored in a stream of bytes. Data can be read in any order and from any position.

- On the other hand, with a serial device, data has to be read as it comes in: byte by byte or in larger pieces. Another type of device is a screen device which is a defined section of the TV or Monitor display itself.

There are also mixtures between all of these types.

The difference between the device types becomes obvious when looking at the operations which can be performed on a device: the files on a drive can be listed in a directory and colours are only available for windows, just to give a few examples. Other operations (especially basic read and write operations) are independent of the device, which is a characteristic of QDOS.

This so-called device independence makes it easy to re-direct basic input or output from a program because the program has no need to know specifics about the device other than its name and/or channel number. If you have Toolkit II installed OPEN supports sub-directories and default devices when used on drives. OPEN will look in the data directory (see DATAD$) for the given file if no device is specified.

Basic details of the various standard devices supported by the QL follow (further details appear in the Drivers Appendix):

| Device Type | Name | Typical uses |
|---|---|---|
| Serial device | ser | Printers, communication with other computers or modems, control of processes, reading analogue data. |
| | par | Output to printers via a centronics interface, |
| | nul | A dummy device which simply receives incoming data and immediately forgets it, useful for debugging. There are several variants available. |
| | pipe | Pipes are intended for communication between jobs, every pipe has an input and output side - there are both standard pipes and named pipes. This is a First In First Out device. |
| | history | Similar to a pipe, except that it is a Last In First Out device. |
| | net | To send or receive data from another network station. |
| | mem | A device to read and write in memory, especially useful to directly access memory on remote network stations via the fileserver. |
| Drives | mdv | Microdrives, the original drives on QLs - files are stored on cartridges. |
| | flp | Floppy disk drives are regarded as standard today - files are stored on disks, early drivers are called fdk. |
| | win | Winchester drives, also called hard disks - files are stored on a permanently installed very large and fast disk. |
| | ram | Ramdisks, virtual but extremely fast drives, the files are stored in RAM and are lost when the computer is switched off. |
| | dev | A kind of universal device, see DEV_USE for an introduction. |
| | pth | Very similar to dev - see PTH_ADD. |
| | mos | Permanent ramdisk, needs specific hardware. |
| | rom | Also a permanent ramdisk. |
| Windows | con | Interfaces to a console device (window) for input and output. |
| | scr | The same as con_ but for output only. |
| Other devices | n | The fileserver device which allows you to access any device on a remote network station. |
| | sdump | A device for a general window dump. |

Please refer to other parts of this book for more specific information on the devices. A lot of examples are given throughout the book.

**NOTE 1**

The OPEN command will close a channel which is already open with the same channel number prior to opening the new channel - do not try to OPEN #0 (except from within a compiled program) unless you have Minerva or SMS - even then, do not try to OPEN #0 as anything other than a CON_ device, except from within a MultiBASIC/Multiple SBASIC.

**NOTE 2**

On AH ROMs, if two tasks tried to read the same file at the same time, the second task was likely to miss the start of the file and read the directory header instead.

**NOTE 3**

On QL ROMs (pre MG) there is a maximum of 32767 OPENs in a session.

**NOTE 4**

The pointer environment has a little bug in it which can lead to odd results when OPENing screen windows. Try, for a laugh (and beware that this will crash the QL eventually), the following:

```
FOR I=1 TO 32768: OPEN #3,scr: PRINT#3,'Hello ';i
```

This is fixed under SMSQ/E and WMAN v1.52.

**NOTE 5**

The maximum number of channels which can be opened at the same time depends on the amount of memory available, but in current implementations, there is an overall maximum of 360 channels, unless you are using Minerva (see below). SMS seems to allow a much larger number of channels to be open at the same time.

**NOTE 6**

Any attempt to open more than one channel to a serial port will report the error 'in use', unless you are using the ST/QL Emulator which allows more than one input channel to be opened to a serial port.

**NOTE 7**

On the QXL (pre v2.50 of SMS), an attempt to OPEN #ch,ser2 would fail if ser1 was not available to the operating system for any reason.

**MINERVA NOTES**

On v1.80 (and later), a third parameter is supported on this command which can be used to specify the 'open type'. This is only of any use on drives and pipes.

**Drives**

| Open type | Effect |
|---|---|
| 0 | Open existing file for exclusive use (same as OPEN) |
| 1 | Open existing file for shared use (same as OPEN_IN) |
| 2 | Open new file (same as OPEN_NEW) |
| 3 | Open file and overwrite if already exists (same as OPEN_OVER) |
| 4 | Open directory file (same as OPEN_DIR) |

(Compare this list with the list at FILE_OPEN!)

**Minerva Example**

```
OPEN#3,ram1_test_device,3
```

opens a new file called ram1_test_device whether or not it already exists.

**Pipes**

The extra parameter supplies the QDOS channel number of the source end of the pipe. This is therefore only of use when opening the 'read' end of the pipe. This gets around the necessity for commands like QLINK. For example these two lines are the same:

```
OPEN#4,'pipe_4000': QLINK#4 TO #3
OPEN#4, 'pipe_4000': pipe_id=PEEK_W (\48\4*40+2) : OPEN#3, 'pipe_', pipe_id
```

Unfortunately, Toolkit II replaces this variant of OPEN with its own, but all of the above facilities (apart from pipe channel numbers) are provided by specific Toolkit II commands in any event. Due to Minerva's System Xtensions, the maximum number of permitted channels open at any one time has been reduced to 304 on an expanded machine (earlier ROMs allow 360). On an unexpanded machine, you can only open 112 under Minerva.

In MultiBasics, both channel #0 and channel #1 can be inextricably linked. Due to the fact that the OPEN command closes an existing channel before setting up the new channel with the given parameters, OPEN #0 or OPEN #1 from within a MultiBasic will remove the MultiBasic in certain instances - see MultiBasic appendix.

**CROSS-REFERENCE**

Opened channels are closed with *CLOSE* and can be listed with *CHANNELS*. *FOPEN* is the same as *OPEN* except it works as a function and *OPEN_IN* / *FOP_IN* open a device for input only. *OPEN_DIR* (*FOP_DIR*) opens a directory (or a sub-directory on level-2 drivers). Also see *OPEN_NEW*, *FOP_OVER*, *TTEOPEN* and *FILE_OPEN*. *NEWCHAN%* can be quite useful when *OPEN*ing channels.

## 22.8 OPEN_DIR

| | |
|---|---|
| Syntax | OPEN_DIR #channel, device_directory or<br>OPEN_DIR #channel, [device_]directory(Toolkit II only) |
| Location | Toolkit II, THOR XVI |

This command is a specialised version of OPEN which is aimed at allowing you to read directories of any given drive device. The directory of a drive contains a copy of every file header which has ever been created on that medium.

When a file is deleted, its entry is blanked out (with zeros) in the directory, thus enabling recovery programs to actually still read the file (provided that nothing else has been written to the sectors where it was stored). It can therefore be very useful to access these directories, for example to provide the user with a selection of files to choose from.

It is however important to differentiate between directories and the output from the DIR command!

On Level-2 and Level-3 device drivers, it is quite easy to access a directory as the directory is stored in a file. For example, on a floppy disk, try:

```
COPY flp1_ TO scr
```

this will show the directory file.

Sub-directories are similar in that after the command:

```
MAKE_DIR flp1_Quill_
```

the file flp1_Quill will be created which contains a copy of all of the file headers for the files within that sub-directory.

Standard device drivers on the other hand are another kettle of fish, in that they allow you to create a file without any name. For example:

```
SAVE mdv1_
```

If you then:

```
COPY mdv1_ TO scr
```

you will see that this is exactly the same as if you had used:

```
SAVE mdv1_boot
```

(apart from the name of the file).

Such files are not revealed by DIR and can be used as a form of copy-protection by some programs. Because of this, you might suffer from a 'Not Found' (-7) error if you tried to:

```
COPY flp1_ TO scr
```

from a disk with a Level-1 device driver. A disk created on a level-1 driver does not look different to a level-2 driver.

If a file with a zero length name was created under a level-1 driver, then this file will only be accessible under the same driver level. To use the command OPEN_DIR, you will need to supply the intended channel number which must be an integer in the range 0…32767. As with OPEN this must be kept as low as possible. After this, comes the name of the directory to be opened. This should generally be simply the name of the device to be accessed, such as:

```
OPEN_DIR #ch,mdv1_
```

OPEN_DIR works correctly with standard device drivers even if there is a file on the drive without a name, eg. mdv1_.

If you have Level-2 device drivers, sub-directories may be accessed by providing the name of the drive plus the name of the sub-directory, for example:

```
OPEN_DIR #3,flp1_Quill
```

If Toolkit II is present, the default data device is supported (see DATAD$), although a directory will still need to be provided, therefore to simply access the default data directory, you will need to use:

```
OPEN_DIR #ch,''
```

Having opened the directory, you can then examine the file header for each file which has been stored on that drive by fetching blocks of 64 bytes from the channel at a time and examining each block per file.

**Example**

A short program which will provide a more detailed directory listing of any device:

```
100 WINDOW 448,200,32,16:PAPER 0:MODE 4:CLS
110 INK 7
120 INPUT 'Read directory of which device? - ';dev$
130 CLS:PRINT 'Directory of ';dev$
140 PRINT 'Filename';TO 40;'File length';TO 54;'Update date'
150 head_start=0
160 INK 4
170 OPEN_DIR #3,dev$:no_files=FLEN(#3)/64
180 FOR listing=1 TO no_files
190   BGET #3\head_start+0,flen1,flen2,flen3,flen4,faccess,ftype
200   flength=flen4+flen3*2^8+flen2*2^16+flen1*2^24-64
210   IF flength>0
220     GET #3\head_start+14, File$
230     BGET #3\head_start+52,fdate1,fdate2,fdate3,fdate4
240     fdate=fdate4+fdate3*2^8+fdate2*2^16+fdate1*2^24
245     IF LEN(File$)=0:File$='<Un-named>'
250     IF ftype<255
260       PRINT File$;TO 40;flength;TO 54;DATE$(fdate)
270     ELSE
280       PRINT File$&'->'
290     END IF
300   END IF
310   head_start=head_start+64
320 END FOR listing
330 CLOSE #3
340 INK 7:PRINT 'End of Listing'
```

**NOTE 1**

The OPEN_DIR command will close a channel which is already open with the same channel number prior to opening the new channel - do not try to OPEN_DIR #0 unless you have read the notes to OPEN!

**NOTE 2**

On QL ROMs (pre MG) there is a maximum of 32767 OPENs (in total) in a session.

**NOTE 3**

If you specify a device which is not actually used for the storage of files (for example:

```
OPEN_DIR#3,scr
OPEN_DIR#3,pipe_1000
```

then this command has exactly the same effect as the OPEN command.

**NOTE 4**

If the specified directory actually points to a non-directory file (or the file does not even exist), then OPEN_DIR will actually open the directory in which that file is located, for example, if the directory flp1_TK_ contained the file flp1_TK_FN_cde:

```
OPEN_DIR#3,flp1_TK_FN_cde
OPEN_DIR#3,flp1_TK_FN
OPEN_DIR#3,flp1_TK
```

would all have exactly the same effect.

**NOTE 5**

Because of the way in which Level-2 and Level-3 device drivers work, provided that you only use the name of an actual directory (or sub-directory) as the parameter, you could actually use OPEN or OPEN_IN instead of OPEN_DIR, but this has its limits, in that it would be useless with standard device drivers and creates havoc if the name of a non-directory file is supplied.

**NOTE 6**

Except under SMS, if a channel has been opened with OPEN_DIR to a main directory, no other channel can access that directory at the same time. Several channels can however be open to the same sub-directory (a bug perhaps) or to a sub-directory further down the tree, which for example allows:

```
100 OPEN_DIR #3,flp1_
110 OPEN_DIR #4,flp1_TK
120 OPEN_DIR #5,flp1_TK
```

but not:

```
100 OPEN_DIR #3,flp1_TK
110 OPEN_DIR #4,flp1_
```

This also has the result that whilst a channel which has been opened with OPEN_DIR is open to a main directory, commands such as DIR, WDIR, WDEL etc. will report 'in use' as they cannot access the directory themselves. The result of this (combined with the operation of the OPEN_DIR command) makes it actually possible to have two channels open to the main directory, by ensuring that the filename passed to the OPEN_DIR commands does not exist on the drive, for example:

```
OPEN_DIR #3,flp1_test
OPEN_DIR #4,flp1_test
```

will leave both channels #3 and #4 open to the main directory (flp1_).

Under SMS you can have several channels open to the same directory thereby avoiding these problems.

**CROSS-REFERENCE**

Please see *OPEN*. Commands such as *FLEN*, *FGETH$* and *HEADR* allow you to examine parts of each files header - see *FGETH$* for details of the file header. *FOP_DIR* is an error trapped version of

*OPEN_DIR*. The Minerva variant of *OPEN*, *OPEN_IN* and *OPEN_NEW* can all be made to work in a similar way to *OPEN_DIR*.

## 22.9 OPEN_IN

| | |
|---|---|
| Syntax | OPEN_IN #channel, device_filename or<br>OPEN_IN #channel, [device_]filename (Toolkit II only) or<br>OPEN_IN #channel, device_filename, type (Minerva v1.80+ only) |
| Location | QL ROM, Toolkit II |

This command is a specialised version of OPEN which is aimed at allowing you to read data from files. This opens the specified channel (#channel must be an integer in the range 0…32767) for input only to the specified filename on the given device.

Any number of channels may be linked to a file using OPEN_IN, although if you try to use any other type of OPEN call to that filename, the error 'in use' will be reported.

The Toolkit II variant of this command supports the default data device if necessary (see DATAD$), but in any case, if the file does not exist (either on the specified device or on the default data device), the error 'Not Found' (-7) will be reported.

**NOTE 1**

OPEN_IN will close a channel which is already open with the same channel number prior to opening the new channel - do not try to OPEN_IN #0 unless you have read the notes to OPEN!

**NOTE 2**

On AH ROMs, if two tasks tried to read the same file at the same time, the second task was likely to miss the start of the file and read the directory header instead.

**NOTE 3**

On QL ROMs (pre MG) there is a maximum of 32767 OPENs in a session.

**NOTE 4**

If instead of device_filename, another type of device is used, such as scr_, OPEN_IN has the same effect as OPEN.

**MINERVA NOTES**

On v1.80 and later, a third parameter is supported by OPEN_IN as with OPEN. This means that this command (if the third parameter is used) has exactly the same effect as OPEN.

**CROSS-REFERENCE**

*FOP_IN* is an error trapped equivalent to this command. *OPEN_DIR* allows you to access directories on drives. *OPEN* contains a general description of all the open types. *OPEN_NEW* and *OPEN_OVER* are also linked with this.

## 22.10 OPEN_NEW

| | |
|---|---|
| Syntax | OPEN_NEW #channel, device_filename or<br>OPEN_NEW #channel, [device_]filename(Toolkit II only) or<br>OPEN_NEW #channel, device_filename, type (Minerva v1.80+ only) |
| Location | QL ROM, Toolkit II |

This command is yet another specialised version of OPEN. This time it is aimed at providing a means of creating a new filename on the specified device and opening a specified channel (#channel must be an integer in the range 0..32767) to that filename for output.

If Toolkit II is present, OPEN_NEW supports the default data device (see DATAD$), however in any case if the device (or default data device) does not contain a formatted medium, the error 'not found' (-7) will be reported. An error will also be reported if the medium is read only.

Without Toolkit II, if the filename already exists, then the error 'already exists' will be generated. On the other hand, Toolkit II will show the familiar 'OK to Overwrite?' prompt.

Once the channel is open, any attempt to open another channel to that same filename at the same time will report an error 'In use'.

**Example**

A simple interactive copying routine:

```
100 INPUT #0,'COPY :-'!in$!'=>'!out$
110 OPEN_IN #3,in$
120 OPEN_NEW #4,out$
130 REPeat copy_loop
140   IF EOF(#3):EXIT copy_loop
150   a$=INKEY$(#3)
160   PRINT a$;:PRINT #4,a$;
170 END REPeat copy_loop
180 CLOSE #4,#3
190 PRINT #0,'Copying complete'
```

**NOTE 1**

The OPEN_NEW command will close a channel which is already open with the same channel number prior to opening the new channel - do not try to OPEN_NEW #0 unless you have read the notes on OPEN!

**NOTE 2**

If instead of device_filename, another type of device is used, such as scr_, OPEN_NEW has the same effect as OPEN.

**NOTE 3**

In version 2.05 of Toolkit II, if the filename already exists, the channel may be left open.

**NOTE 4**

Similar problems exist with OPEN_NEW to those encountered with SAVE when trying to write to a write-protected microdrive cartridge. Unfortunately however, the problem is made worse by the fact

that the problem is not revealed when the channel is opened. Instead 'bad or changed medium' is only displayed when the file is written to (ie. when 512 characters have been written to the channel, or the channel is CLOSEd).

**CROSS-REFERENCE**

*FOP_NEW* is an error trapped function which is equivalent to this command. *OPEN_DIR* allows you to access directories on drives. *OPEN* contains a general description of all the open types. *OPEN_IN* and *OPEN_OVER* are also linked with this.

## 22.11 OPEN_OVER

| | |
|---|---|
| Syntax | OPEN_OVER #channel, device_filename or<br>OPEN_OVER#channel, [device_]filename(Toolkit II only) |
| Location | Toolkit II, THOR XVI |

This command is exactly the same as the Toolkit II version of OPEN_NEW except that if the specified filename already exists, the filename is automatically overwritten. Also, the THOR XVI version of this command does not support the default data device.

**CROSS-REFERENCE**

See *OPEN_NEW*! The Minerva variant of *OPEN*, *OPEN_IN* and *OPEN_NEW* can all be made to work in the same way as *OPEN_OVER*. *FOP_OVER* is a function which operates like *OPEN_OVER* except that it allows any errors to be trapped.

## 22.12 OR

| | |
|---|---|
| Syntax | condition1 OR condition2 |
| Location | QL ROM |

This combination operator combines two condition tests together and will have the value 0 if both condition1 and condition2 are false, or 1 if either condition1 or condition2 are true (or both are true). Please note the difference between this and the bitwise OR operator: x||y, which compares x and y bit by bit.

**Example 1**

```
PRINT 1 OR 0
```

Returns 1.

```
PRINT 12 OR 10
```

Returns 1.

Compare PRINT 12&&10 which returns 14).

**Example 2**

```
10 FOR x=1 TO 5
20   FOR y=1 TO 5
30     IF x=3 OR y>3:PRINT x;'=>';y,
40   END FOR y
50 END FOR x
```

produces the following output:

```
1=>4 1=>5 2=>4 2=>5 3=>1 3=>2 3=>3 3=>4 3=>5 4=>4 4=>5 5=>4 5=>5
```

**CROSS-REFERENCE**

*AND*, *NOT* and *XOR* are the other combination operators.

## 22.13 OUTL

| | |
|---|---|
| Syntax | OUTL [#]chan [,width,height,x,y] |
| Location | PEX |

This command is similar to OUTLN except for a few variations:

1. If chan is not preceded by # then it is taken to be a QDOS channel number (and this command can therefore be used to redefine an Outline for any Job).
2. You cannot specify a shadow.
3. If only the chan parameter is used (with or without a #), then the current maximum sizes of the Jobs windows are used (similar to *OUTLN* without any parameters).

**CROSS-REFERENCE**

See *OUTLN*. *CHANNELS* allows you to find out about QDOS channel numbers.

## 22.14 OUTLN

| | |
|---|---|
| Syntax | OUTLN [#chan,] width,height,x,y [,x_shad,y_shad] or<br>OUTLN (SMSQ/E only) |
| Location | ATARI_REXT (v2.12+), SMSQ/E |

This command is used within the Pointer Environment to signal that a specified window (default #0) which must already be open, is to be looked after by the Pointer Environment (managed).

Because of the way in which the Pointer Environment works, it is always a good idea to use OUTLN on the first window to be used for input/output by a program (this is known as the Primary Window), as this will ensure that all windows which are subsequently OPENed by the program will be what is known as Secondary Windows and also managed.

Because of this, if a program is to be run under the SuperBASIC interpreter, OUTLN should be used on #0, whereas in a compiled program, OUTLN needs to be used on the first channel which is OPENed (ensure that the program is compiled without any windows open).

Hints on writing programs to run under the Pointer Environment appear below, showing how OUTLN should be used.

If an OUTLN has been defined, any attempt to OPEN a window which would fall outside of the managed Primary Window will cause an 'out of range' error. If you then use OUTLN on a Secondary window, the first time that OUTLN is encountered after the window is OPENed, the contents of the screen under that window will be stored. Then, if you again use OUTLN on the same window, the contents of the screen under the Secondary Window are restored (see the example).

With the first syntax of the command, the first five parameters supplied to OUTLN are exactly the same as with WINDOW, however, you can also add two further parameters, x_shad and y_shad to specify the width of a shadow which will appear to the right and bottom (respectively) of the window to make it stand out. They both default to zero (no shadow).

SMSQ/E v2.53+ allows the second syntax, which will allow you to use OUTLN without any parameters at all. In this case, the primary window will be outlined to the smallest area which can encompass all currently OPEN windows at the time that OUTLN is used.

**Writing programs to use the Pointer Environment**

Some information concerning this appears in Section 4, however, when designing a program to use the Pointer Environment, it is useful to follow this procedure:

1. Open a main channel to define the maximum screen area available to the job, eg: OPEN #1,con_ This should be the first window OPENed by the program - if it is compiled, compile the program without Windows enabled.

2. Fetch the screen limits, eg:

```
scr_width%=QFLIM (#1,0)
scr_height%=QFLIM (#1,1)
```

3. Ensure that the screen is in the right mode:

```
IF RMODE<>0: MODE 4
```

4. Outline #1 (the main channel) to the size of the program:

```
OUTLN #1,450,210,0,0
```

The program will then have a maximum screen area of 450x210 pixels available. When you wish to resize the program's display, you will need to mark the main channel (#1) as unmanaged and then use OUTLN to resize the main channel. For example, the following method was used (using commands from EasyPTR by Jochen Merz Software) to allow the user to re- size the program Q-Route (available from Q Branch):

The procedure is called when the user highlights the Resize Loose Item on the main menu (which is drawn on #1). In order for this to work, the main menu had to be loaded as a separate file into the common heap area pointed to by the variable m_store (as there is no way in current versions of EasyPTR to allow you to find the address of the original menu definition in an Appended definition file - this is not the working menu definition used by the Window Manager).

For more general information on EasyPTR, you are directed to the EasyPTR tutorial contained in the Quanta magazine in 1994. The outline of a routine (excuse the pun) to re-size the main menu used by a program appears on the next page (note that this requires EasyPTR (c) Albin Hessler, and substantial additions to the code in order to work):

```
9620 DEFine PROCedure RESIZE_MAIN
9621   sel_key%=0
9622   DIM result%(16)
9630   PVAL #Main_menu,result%
9635   old_x%=result%(14):old_y%=result%(15)
9637   : REMark Fetch original pointer co-ordinates
9640   pxpos%=old_x%:pypos%=old_y%
9650   RDPT #Main_menu,130,pxpos%,pypos%
9651   : REMark Draw and move re-size ICON
9652   : REMark NOTE THIS CRASHES SUPERBASIC!!
9655   PVAL #Main_menu,result%
9660   IF result%(6)=27:st%=MSTAT%(#Main_menu,-3,0):RETurn:
9662   : REMark ESC pressed therefore ignore new setting
9665   Menu_add=m_store
9667   : REMark Look at where original Menu definition is stored.
9670   pwidth=PEEK_W(Menu_add+28):pheight=PEEK_W(Menu_add+30)
9675   : REMark These offsets contain the size of the existing menu
9675   px=prog_x:py=prog_y
9685   pwidth=pwidth-(pxpos%-old_x%):IF pwidth MOD 2:pwidth=pwidth+1
9690   pwidth=MAX(pwidth,450)
9695   pwidth=MIN(pwidth,scr_width%-12)
9700   px=MIN(pxpos%-34,(scr_width%-pwidth)-12)
9705   px=MAX(px,0)
9710   pheight=MAX(pheight-(pypos%-old_y%),210)
9715   pheight=MIN(pheight,scr_height%-10)
9720   py=MIN(pypos%-5,(scr_height%-pheight)-10)
9725   py=MAX(py,0)
9726   : REMark the lines 9675-9725 calculate the new width and height
9727   : REMark of the menu (minimum size 450x210)
9728   : REMark (maximum size scr_width%-12 x scr_height%-10)
9755   prog_x=px:prog_y=py
9760   MCLEAR #Main_menu:CLPT #1
9762   : REMark Remove the old working menu definition
9765   OUTL #1,pwidth,pheight,px,py
9770   : REMark Resize outline & main window dimensions
9775   POKE_W Menu_add+28,pwidth:POKE_W Menu_add+30,pheight
9780   POKE_W Menu_add+76,pwidth:POKE_W Menu_add+78,pheight
9782   : REMark Alter the menu sizes in the menu definition
9784   :
9785   : REMark You will now need to re-position various loose items as␣
↪necessary
9787   : REMark There is no easy way to find the offsets of the definitions
9788   : REMark within the original menu definition.
9790   : REMark You will also need to re-size Information and Application Sub-
↪Windows
9795   : REMark as necessary.
9795   :
9865   MDRAW #1,m_store,px,py:Main_menu=MWDEF(#1)
9866   : REMark Redraw the main menu, creating a new Working Menu Definition
9870   : REMark you will now need to redraw any information which is normally␣
```

(continues on next page)

(continued from previous page)

```
↪shown in the
9875   : REMark main menu but not contained in the menu when it was designed.
9885 END DEFine
```

**Example**

A short program which produces a graphical effect and then provides a pull-down menu on a secondary window, using OUTLN to restore the screen after you have used the menu.

```
100 OUTLN #0,512,256,0,0
110 PAPER #0,0:CLS#0
120 REMark Force #0 to Primary Window
130 WINDOW #0,448,40,32,216
140 WINDOW 448,200,32,16
150 PAPER 2:INK 7:CLS
180 PRINT 'This is a Secondary Window'
190 REPeat loop
200   INK RND(3 TO 7)
210   FOR i=0 TO 360 STEP RND(10 TO 30)
220     x=RAD(i):LINE 50,50 TO 50-40*SIN(x),50-40*COS(x)
230   END FOR i
235   OPEN #3,scr_400x100a56x20
236   PAPER #3,0:INK #3,7
240   OUTLN #3,400,100,56,20:CLS#3
250   PRINT #3,' MENU'
260   PRINT #3,'Press <ESC> to leave'
270   PRINT #3,'Press <SPACE> to continue'
280   REPeat keys
290     x$=INKEY$(-1):IF x$ INSTR ' '&CHR$(27):EXIT keys
300   END REPeat keys
310   OUTLN #3,400,100,56,20
315   CLOSE #3
320   IF x$=CHR$(27):EXIT loop
330 END REPeat loop
```

Note the need to CLOSE #3 each time that it is removed from the screen.

If #3 was OPENed outside of the loop, OUTLN would only save the contents of the screen under #3 the first time that line 240 was encountered, and each subsequent time that it was used, will try to restore the contents of the screen!

**NOTE 1**

If you use OUTLN to reduce the area of a Primary Window, any Secondary Windows which would contain an area outside of the new Primary Window will be re-sized so that they have exactly the same size and position as the new Primary Window. Any saved contents will be lost. This is also true of any windows which are OPENed after an OUTLN command - if they would fall outside of the area defined by OUTLN, then the newly OPENed window will occupy the same area as the OUTLN. Compare WINDOW which will cause an error.

**NOTE 2**

Before v2.58 of SMSQ/E, OUTLN without any parameters did not work if an OUTLN was already set.

**CROSS-REFERENCE**

See *QFLIM*. *OUTL* is similar. *WMON* and *WTV* also add an outline to a program.

## 22.15 OVER

| | |
|---|---|
| Syntax | OVER [#channel,] switch |
| Location | QL ROM |

This command allows you to set the way in which anything is written to a specified window (default #1), whether by PRINT, LINE, BLOCK, or any other command which prints something on a window. If the supplied channel is not a window, then error -15 (bad parameter) will be generated, as will any value of switch outside of the range -1..1. When the QL is first initiated (or following a MODE command), OVER is set to 0 (see below). This can be altered by giving a different value for switch which will have the following effect:

| Switch | Effect |
|---|---|
| -1 | Everything is PRINTed on a transparent strip. However, each pixel which is drawn on that window in the current INK (or with BLOCK) is actually xored with the colour of the existing background. |
| 0 | This is the standard mode, where characters are PRINTed in the current INK and STRIP and any pixels plotted on screen are also in the current INK. |
| 1 | This PRINTs characters on a transparent STRIP but pixels are drawn in the current INK colour. BLOCK is unaffected. |

When OVER -1 is used, it may be useful to calculate how different colours will appear on screen. This can be achieved by XORing the two colours in binary, with col1 ^^ col2, for example, a line drawn in blue on a white background with OVER -1 will actually appear on screen to be INK 1^^7=6 (Yellow). A result of OVER -1 is that if something is drawn twice in the same place in the same colour, the object is effectively removed from the screen, leaving the screen unaltered. This can be seen in the example program given for IF.

**Example**

A simple demonstration which shows the effects of OVER on CIRCLE, PRINT and BLOCK. See how easy/difficult it is to calculate how the end display will look:

```
100 MODE 8:WINDOW 448,200,32,16:PAPER 0:CLS
110 INK 2:SCALE 100,0,0
120 FILL 1:CIRCLE 50,50,35
125 PAUSE
130 INK 7:OVER -1:FILL 1:CIRCLE 50,50,35
135 PAUSE 140 FILL 0:OVER 0
150 PAPER 4:INK 7:PRINT"This is a simple circle"
155 PAUSE
160 OVER 1:PRINT"This is another line of text"
165 PAUSE
170 OVER -1:PRINT\"This is yet another line"
175 PAUSE
180 BLOCK 448,200,0,0,2
```

**NOTE 1**

OVER 0 is set after a MODE command.

**NOTE 2**

The following appears to be a bug in Minerva (pre v1.96) and most other implementations:

On Minerva pre v1.96, OVER#0 and OVER#1 are equivalent to OVER#1,0 and OVER#1,1 respectively, OVER#2 gives bad parameter, as does OVER#-1. OVER is equivalent to OVER #1,0!!

On all later versions of Minerva and SMS, the behaviour is more logical:

The channel number defaults to #1 and the switch to 0, so OVER#0 is OVER#0,0, OVER#1 is OVER#1,0 (not OVER#1,1), OVER#2 is OVER#2,0 and OVER#-1 naturally produces a 'channel not open' error.

**NOTE 3**

OVER -1 causes various problems with the FILL command - see FILL.

**CROSS-REFERENCE**

Please look at *INK* and *PRINT*.

CHAPTER

# TWENTYTHREE

# KEYWORDS P

## 23.1 PAGDIS

| | |
|---|---|
| Syntax | PAGDIS (lines) lines=0..127 |
| Location | Beuletools |

This function returns the printer control codes needed to set the length of the page header when sent to EPSON compatible printers.

```
PRINT PAGDIS (lines)
```

is equivalent to:

```
PRINT CHR$(27) & CHR$(78) & CHR$(lines)
```

**Example**

To set the header to three lines:

```
OPEN #3,ser1:PRINT #3,PAGDIS(3):CLOSE#3
```

**CROSS-REFERENCE**

*PAGLIN*, *PAGLEN*, *LMAR*, *RMAR*.

## 23.2 PAGLEN

| | |
|---|---|
| Syntax | PAGLEN (inches) inches=0..22 |
| Location | Beuletools |

This function returns the control codes needed to set the length of a page (in inches) when sent to EPSON compatible printers. This is normally 12”. The function is equivalent to:

```
CHR$(27) & 'C' & CHR$(0) & CHR$(inches)
```

**CROSS-REFERENCE**

*PAGDIS*, *PAGLIN*, *LMAR*, *RMAR*.

## 23.3 PAGLIN

| | |
|---|---|
| Syntax | PAGLIN (lines) lines=0..127 |
| Location | Beuletools |

This function returns the control codes needed to set the length of a page (in lines) when sent to EPSON or compatible printers. This is normally either 66 or 72 lines.

```
PRINT PAGLIN (lines)
```

is the same as:

```
PRINT CHR$(27) & 'C' & CHR$(lines)
```

**CROSS-REFERENCE**

*PAGDIS*, *PAGLEN*, *LMAR*, *RMAR*.

## 23.4 PAINT

| | |
|---|---|
| Syntax | PAINT x, y, col, bufadr, buflen |
| Location | HCO |

The command PAINT addresses the screen directly and fills a closed figure with the colour col (which must be in the range 0..4, see SET).

The command requires a buffer of at least 4K whose start address is passed to PAINT as bufadr and whose length as buflen (which must be a minimum of 4096 bytes). The larger the buffer, the better, but very large buffers (say, 100K) seem to confuse PAINT and will make it stop before it has finished - therefore keep the buffer below 32K.

**Example 1**

Random drawing:

```
100 WINDOW 512,256,0,0: BORDER 1,3: PAPER 0: CLS
110 buflen = 10240
120 :
130 FOR i = 1 TO 15
140   LDRAW RND(511),RND(255) TO RND(511),RND(255), 3
150 END FOR i
160 :
170 buffer = ALCHP(buflen)
180 PAINT RND(1 TO 510), RND(1 TO 255), 2, buffer, buflen
190 RECHP buffer
```

**Example 2**

A spectacular crash!

```
PAINT 50,50,4,131072,32768
```

**WARNING 1**

See those at SET.

**WARNING 2**

PAINT will crash the machine if it runs out of the screen, so you have to ensure that the figure to be filled is closed. Another means of protection is to use a BORDER, as in the example.

**CROSS-REFERENCE**

*FILL*, *SET*, *LDRAW*

## 23.5 PALETTE_QL

| | |
|---|---|
| Syntax | PALETTE_QL [#ch,] start, true_colour1 *[,true_colourx]* |
| Location | SMSQ/E v2.98+ |

This command allows you to redefine the eight colours used by the Extended Colour Drivers to display COLOUR_QL. A valid window channel must be open, default #1 (or #0 on a SBASIC with only #0 open), although one may also be supplied as #ch.

Start is the number of the first colour to change, followed by each of the new colours described in 24 Bit Colour Mode. On hardware with a true palette map (most PCs), this command will affect all programs, including information already displayed on screen. However, on all other hardware, most notably the Q40 and Q60, existing information will remain unaffected.

**Example 1**

```
PALETTE_QL 4,$FFB6DB
```

makes the computer use PINK instead of GREEN when INK 4 (QL Colour Value) is used within a program.

```
PALETTE_QL 5,$B6FFFF,$929200
```

will change INK 5 to Light Blue (from Cyan) and INK 6 to Mustard (from Yellow).

**Example 2**

Many programs written with MODE 4 in mind, presume that INK 3 is the same as INK 2 (for example). However, under COLOUR_QL, even MODE 4 programs can access the standard MODE 8 colours, therefore INK 3 becomes MAGENTA. To overcome this problem, use the following routine:

```
100 red=255*65536+100:REMark $FF0064 - red and a bit of blue
110 blue=255*256+155: REMark $00FF9B - green and the rest of blue
120 white=blue+red:   REMark $FFFFFF
125 REMark - Now change all 8 colours, starting at INK 0
130 PALETTE_QL 0,0,0,red,red,blue,blue,white,white
```

**NOTE**

The problem with using 24 bit values is that they have to be trimmed to fit into the native colour scheme on the computer in use - as a result, although the same 24 bit colour value is specified, the resultant colour will be slightly different on QPC, Q40 and Aurora.

**CROSS-REFERENCE**

*COLOUR_QL*, *INK*, *PALETTE_8* Also refer to Appendix 16 which lists the first 256 24 Bit Colours.

## 23.6 PALETTE_8

| | |
|---|---|
| Syntax | PALETTE_8 [#ch,] start, true_colour1 *[,true_colourx]* |
| Location | SMSQ/E v2.98+ |

This command is similar to PALETTE_QL, except that it allows you to redefine all 256 colours available under COLOUR_PAL.

As with PALETTE_QL, start is the number of the first colour to change, followed by each of the new colours described in 24 Bit Colour Mode. A valid window channel must also be open, default #1 (or #0 on a SBASIC with only #0 open), although one may also be supplied as #ch.

On hardware with a true palette map (most PCs), this command will affect all programs, including information already displayed on screen. However, on all other hardware, most notably the Q40 and Q60, existing information will remain unaffected.

**Examples**

```
PALETTE_8 4, $FFB6DB
```

makes the computer use PINK, instead of BLUE when INK 4 (PAL Colour Value) is used within a program.

```
PALETTE_8 5, $B6FFFF, $929200
```

will change INK 5 to Light Blue (from Magenta) and INK 6 to Mustard (from Yellow).

**NOTE 1**

This command will not have any effect on Aurora, which only provides 256 colours to choose from. It therefore *may not* be implemented on the Aurora version of SMSQ/E.

**NOTE 2**

The problem with using 24 bit values is that they have to be trimmed to fit into the native colour scheme on the computer in use - as a result, although the same 24 bit colour value is specified, the resultant colour will be slightly different on QPC and Q40.

**CROSS-REFERENCE**

See *PALETTE_QL* and *COLOUR_PAL* for more details.

## 23.7 PAN

| | |
|---|---|
| Syntax | PAN [#ch,] distance [,area] |
| Location | QL ROM |

This command is very similar to SCROLL except that this enables you to move a window left and right as opposed to up and down. In its most simple form, PAN allows you to move the specified window (default #1) a given number of pixels sideways. If a positive value for the distance is given, the window will move to the right, whereas a negative distance will move the window to the left. Again, as with SCROLL, the gap left behind from the move will be coloured in the current PAPER colour, and any contents of the window moved off the screen will be lost. The additional parameter allowed by PAN lets you specify an area of the given window to be moved. This can have one of the following values:

| Area | Effect |
|---|---|
| 0 | This moves the whole window (this is the default). |
| 3 | This moves the whole of the text cursor line. |
| 4 | This moves everything on the text cursor line to the right of the cursor (including the character under the cursor). |

**Example**

A short procedure to scroll a given text message across the screen:

```
100 DEFine PROCedure SCROLL_TEXT(line$)
110   LOCal l$,loop
120   l$=line$
130   OPEN #3,scr_448x10a32x246
140   PAPER#3,2:INK#3,0:CSIZE#3,1,0:CLS#3
150   AT #3,0,0:PRINT#3,'INCOMING MESSAGE:'
160   INK#3,7
170   REPeat loop
180     IF LEN(l$)=0:EXIT loop
190     AT #3,0,55:PRINT#3,l$(1)
200     BEEP 100,10
210     IF LEN(l$)<=1:EXIT loop
220     l$=l$(2 TO)
230     AT #3,0,18:PAN #3,-8,4
240     PAUSE 30
250   END REPeat loop
260 END DEFine
```

**NOTE 1**

In low-resolution mode (8 or 12), the distance will always be rounded down to an even number of pixels.

**NOTE 2**

As with SCROLL, odd values for area and distance allow you to access machine code routines (unless you have a THOR XVI, SMS, or Minerva ROM (v1.63 or v1.64). To access these machine code routines:

- Take the TRAP #3 value for D0 and deduct 27.
- If the figure is less than 27, then take the negative result and add to 128.

For example, PAN 0,115 turns on cursor in #1 (TRAP #3,D0=$E).

This is in fact more useful than SCROLL or CLS as area can be used to pass a parameter to D1, thus allowing you for example, to access IOF.POSR - use PAN #3,n%,40 - to move the file pointer.

**NOTE 3**

On pre MG ROMs, this command may fail if the window is smaller than the cursor.

**NOTE 4**

On SMSQ/E pre v2.88, PAN in MODE 8 could ruin the screen display if an odd parameter was specified, since SMSQ/E tried to move the screen by an odd number of pixels (not supported in MODE 8). Although annoying, this only had small effects.

**CROSS-REFERENCE**

Also please see *SCROLL* and *PAPER*. THORs allow you to use *IO_TRAP* to access additional system calls. Most system calls can be accessed using Toolkit II in any event. Otherwise, see *BTRAP*, *QTRAP*, *TTET3* and *MTRAP*. The QDOS/SMS Reference Manual Section 15 contains full details of the TRAP #3 calls.

## 23.8 PAPER

| | |
|---|---|
| Syntax | PAPER [#window,] colour or<br>PAPER [#window,] colour1,colour2 [,pattern] |
| Location | QL ROM |

This command sets the background colour inside a window (default #1). Characters printed to that window will be written with the PAPER colour as a background unless another colour has been specified with STRIP.

**Example**

```
100 OPEN#3,scr_512x256a0x0
110 REPeat forever
120   FOR c=0 TO 7
130     BORDER#3,RND(100)
140     PAPER#3,c
150     CLS#3
160   END FOR c
170 END REPeat forever
```

**NOTE**

PAPER also resets the STRIP to the specified colour.

**CROSS-REFERENCE**

*INK* sets the foreground colour and *STRIP* the background for characters only. *CLS* clears a window in the current paper colour. See *INK* concerning colour in general.

## 23.9 PARHASH

| | |
|---|---|
| Syntax | PARHASH (parameter) |
| Location | PARAMS (DIY Toolkit - Vol P) |

This is an addition to the normal PARUSE and PARNAM$ functions which allows you to check whether a value passed as a parameter to a SuperBASIC PROCedure or FuNction was preceded by a hash or not.

**Example**

The following PROCedure allows you to create a CAT command which is similar to DIR, allowing you to use the following syntaxes:

```
CAT #channel [,device]
CAT [#channel,] [device]
```

to read a directory.

If device is not specified, then a directory of the default data device is produced. If a channel is not specified, then #1 will be used. The device can be in quotes or not if you prefer. The following can therefore all be used:

```
CAT #2
CAT CAT flp1_
CAT #3,'win1_'
```

```
100 DEFine PROCedure CAT (ch,direct)
110   LOCal dir_ch,sepa%,hash%
112   hash%=PARHASH(ch): sepa%=PARSEPA(ch)
120   IF sepa%>0
130     file$=PARSTR$(direct,2)
140   ELSE
150     IF hash%
160       file$=DATAD$
170     ELSE
180       file$=PARSTR$(ch,1):ch=1
185       IF file$='': file$=DATAD$
187     END IF
190   END IF
200   dir_ch=FOP_DIR(file$)
210   IF dir_ch<0: PRINT #0,'CANNOT ACCESS DIRECTORY ON ';file$:RETurn
220   CLOSE #dir_ch
230   DIR #ch,file$
250 END DEFine
```

**NOTE 1**

There is a problem with this function that prevents the above example from working under SMS - once either PARHASH or PARSEPA have been used once on a parameter, they will not work again!!

For example, try adding the following lines to the above and compare the results:

```
116 PRINT PARHASH(ch), PARSEPA(direct), PARNAME$(2), PARTYP(ch),␣
↪PARTYPE(direct)
117 PRINT PARHASH(ch), PARSEPA(direct), PARNAME$(2), PARTYP(ch),␣
↪PARTYPE(direct)
118 STOP
```

**NOTE 2**

TURBO and SuperCHARGE cannot compile programs which use PARHASH.

**CROSS-REFERENCE**

*PARTYP*E, *UNSET* and *PARNAME$* are also added by this toolkit.

## 23.10 PARNAM$

| | |
|---|---|
| Syntax | PARNAM$ (number) |
| Location | Toolkit II |

This function can be used to find the name of an actual parameter passed to a SuperBASIC PROCedure or FuNction. You merely need to supply the number of the parameter in the definition line which you wish to find. If the parameter was passed as a name (ie. by reference), then this name will be returned by PARNAM$, however, in all other cases, a nul string will be returned.

**Example**

A short procedure which prints the square of the parameter passed (and if possible squares the actual parameter!):

```
1000 DEFine PROCedure Square (x)
1010   LOCal param$,loop,key$
1020   param$=PARNAM$(1)
1030   IF param$<>'' THEN
1040     PRINT #0,param$!'will be altered - is this okay?'
1050     REPeat loop:key$=INKEY$(-1):IF key$ INSTR 'yn':EXIT loop
1060     IF key$=='n':RETurn
1070   END IF
1080   x=x^2:PRINT x
1090 END DEFine
```

Compare the following:

```
number=2:Square number: REMark number is passed by reference
number=2:Square (number): REMark number is passed by value
```

**NOTE**

TURBO and SuperCHARGE cannot compile programs which use PARNAM$.

**CROSS-REFERENCE**

*PARTYP*, *PARUSE* and *PARSTR$* allow you to find out other information about parameters. See also *DEFine FuNction* and *DEFine PROCedure*. *PARNAME$* is exactly the same.

## 23.11 PARNAME$

| | |
|---|---|
| Syntax | PARNAME$ (number) |
| Location | PARAMS (DIY Toolkit - Vol P) |

This is exactly the same as PARNAM$.

**CROSS-REFERENCE**

*PARTYP*E, *PARHASH* and *PARSEPA* are also added by this toolkit.

## 23.12 PARSEPA

| | |
|---|---|
| Syntax | PARSEPA (name) |
| Location | PARAMS (DIY Toolkit - Vol P) |

This function is a useful addition that allows you to check on the type of separator which follows a value passed as a parameter to a SuperBASIC PROCedure or FuNction. The value returned by PARSEPA is:

| PARSEPA | Meaning |
|---|---|
| 0 | No separator follows - this is the end of the line. |
| 1 | A comma (,) follows. |
| 2 | A semicolon (;) follows. |
| 3 | A backslash (\) follows. |
| 4 | An exclamation mark (!) follows. |
| 5 | The word TO follows. |

**NOTE**

This function suffers from the same problems as PARHASH.

**CROSS-REFERENCE**

See*PARHASH* in particular - this contains an example which uses this function.

## 23.13 PARSTR$

| | |
|---|---|
| Syntax | PARSTR$ (name,number) |
| Location | Toolkit II |

This function, together with its associated functions: PARTYP, PARUSE and PARNAM$ allows you to find out information about a parameter passed to a SuperBASIC PROCedure or FuNction.

PARSTR$ is aimed for use in PROCedures or FuNctions where a user might more naturally pass a parameter as a name rather than a string (for example, when passing a filename).

Many users will have noted how many machine code procedures and functions do not need filenames to be passed as a string, for example:

```
SAVE flp1_boot
```

and wondered why when they write a PROCedure, it has to be passed as a string, for example:

```
SSAVE 'flp1_boot'
```

Well, PARSTR$ allows you to do either!!

The two parameters which need to be supplied to PARSTR$ are the name of the parameter as listed in the definition line and the number of that parameter in the parameter list.

**Example**

A re-write of a SAVE routine to keep the current file version up to date (this could be used for example when developing a program):

```
100 DEFine PROCedure SSAVE(file)
110   LOCal version
120   file$=PARSTR$(file,1)
130   version=FVERS(\file$)
140   SAVE file$
150   SET_FVERS \file$, version+1
160 END DEFine
```

To update the saved version of the program in memory, you can then use either:

```
SSAVE flp1_program_bas
```

or

SSAVE 'flp1_program_bas'.

Note that SMS users can just use SAVE (without a filename) to achieve this.

**NOTE 1**

If you try to assign the string returned by PARSTR$ back into the original parameter (eg. change the variable file in the example program to the variable file$), this will cause an 'error in expression'. You could try adding file$ to the LOCal definition, however although this will avoid the 'error in expression', file$ is set to a nul string by the LOCal definition!!

**NOTE 2**

TURBO and SuperCHARGE cannot compile programs which use PARSTR$.

**CROSS-REFERENCE**

Please also see *PARNAM$*. *FBKDT* also contains a useful example of *PARSTR$*.

## 23.14 PARTYP

| | |
|---|---|
| Syntax | PARTYP (name) |
| Location | Toolkit II, THOR XVI |

As disclosed in the description of DEFine FuNction, a parameter is passed to a SuperBASIC PROCedure or FuNction by reference, meaning that the variable type listed in the definition line will actually be overriden by the type of variable which has been passed as a parameter.

The function PARTYP returns the type of the actual parameter which has been passed, which can be used to error trap PROCedures and FuNctions. PARTYP expects only one parameter - the name of the parameter from the definition line to be looked at. PARTYP will then return one of the following values depending on the type of the actual parameter passed:

| PARTYP | Meaning |
|---|---|
| 0 | A null string has been passed. |
| 1 | A string has been passed. |
| 2 | A floating point has been passed. |
| 3 | An integer has been passed. |

**Example**

A PROCedure to swap any two variables (it can only handle simple strings and variables, not arrays):

```
100 a=1:b%=2
110 swap_var a,b%
115 :
120 DEFine PROCedure swap_var (x,y)
130   LOCal xa,xa$,param
140   IF PARUSE(x)=0 OR PARUSE(y)=0
150     PRINT 'A variable is unset!':RETurn
160   END IF
162   IF PARNAM$(1)="" OR PARNAM$(2)=""
164     PRINT 'Parameters are not both variables!':RETurn
165   END IF
170   IF PARUSE(x)=3 OR PARUSE(y)=3
180     PRINT 'Arrays not handled':RETurn
190   END IF
200   param=PARTYP(x)
210   IF PARTYP(y)=1 AND param<>1 OR param=1 AND PARTYP(y)<>1
220     PRINT 'You cannot swap a string with a value!'
230     RETurn
240   END IF
250   SELect ON param
260     =0,1:xa$=y:y=x:x=xa$
270     =2,3:xa=y:y=x:x=xa
280   END SELect
290 END DEFine
```

**NOTE 1**

There is a difference in the way that PARTYP will report an omitted parameter, depending on whether you implicitly omit the parameter. Try the following using the above example:

Implicit omission:

```
swap_var a$
```

or even:

```
swap_var a$,
```

PARTYP(y) returns the type of the notional parameter (here y is a floating point, so PARTYP (y) returns 2), and PARUSE(y) also reports 2. Compare explicit omission:

```
swap_var ,a$
```

PARTYP(x) will return 0 as will PARUSE(x) If a program is Qliberated, PARTYP will return 0 whether parameters are implicitly or explicitly omitted.

**NOTE 2**

If you pass a null string as a parameter, eg:

```
swap_var a$,''
```

in the above example, PARTYP will still return 1 (and not zero) as you may think, hence the need to look at PARUSE also.

**NOTE 3**

TURBO and SuperCHARGE cannot compile programs which use PARUSE.

**CROSS-REFERENCE**

*PARTYP* should be used alongside *PARUSE* to find out whether a parameter was passed as a variable (ie. by reference) or as a value. *PARTYPE* is the same. *PARNAM$*, *PARHASH*, *PARSEPA* and *PARSTR$* form companions to these commands.

## 23.15 PARTYPE

| | |
|---|---|
| Syntax | PARTYPE (name) |
| Location | PARAMS (DIY Toolkit - Vol P) |

This function is exactly the same as PARTYP and suffers from the same problems.

**CROSS-REFERENCE**

*PARHASH*, *UNSET* and *PARNAME$* are also added by this toolkit.

## 23.16 PARUSE

| | |
|---|---|
| Syntax | PARUSE (name) |
| Location | Toolkit II, THOR XVI |

PARUSE is a companion function to PARTYP. PARUSE also expects only one parameter - the name of the parameter from the definition line to be looked at. PARUSE will then return one of the following values depending on the actual parameter passed:

| PARUSE | Meaning |
|---|---|
| 0 | An unset variable has been passed. |
| 2 | A variable (or value) has been passed. |
| 3 | An array has been passed. |

**NOTE 1**

The Toolkit II Manual gives incorrect values.

**NOTE 2**

There is a difference in the way that PARUSE will report an omitted parameter, depending on whether you implicitly omit the parameter or explicitly omit it - see Note 1 relating to PARTYP. Under current versions of Qliberator a program, PARUSE will always return 2 whether the parameter is implicitly or explicitly omitted.

**NOTE 3**

TURBO and SuperCHARGE cannot compile programs which use PARUSE.

**CROSS-REFERENCE**

Please see *PARTYP*.

## 23.17 PAR_ABORT

| | |
|---|---|
| Syntax | PAR_ABORT or<br>PAR_ABORT port_number(SMSQ/E only) |
| Location | ST/QL, SMSQ/E |

This command clears out all of the closed PAR buffers and then sends an 'aborted' message, to the PAR device, thus effectively stopping the computer from sending any information still in the buffers through the PAR device. Any open channels connected to the port are unaffected.

**NOTE**

Although the SMSQ/E implementation allows a port to be specified, there are currently no implementations of the QL which have more than one parallel port, therefore trying to pass a port_number at present results in a bad parameter error.

**CROSS-REFERENCE**

*SER_ABORT* and *PRT_ABORT* are very similar. *PAR_CLEAR* clears out the buffers but does not tell anyone. *PRT_ABT* is similar on the Trump Card and Gold Cards.

## 23.18 PAR_BUFF

| | |
|---|---|
| Syntax | PAR_BUFF [size] or<br>PAR_BUFF port_number, size(SMSQ/E only) |
| Location | ST/QL, SMSQ/E |

Normally, SMSQ/E and the Emulator will use all available memory as a buffer for its serial and parallel ports (this is known as a dynamic buffer). Although this enables control to be returned to programs very quickly after sending output to one of the ports, it can however mean that the whole of the memory can be filled up with printer output.

The command PAR_BUFF therefore allows you to specify a fixed size in bytes for the parallel buffer for each channel opened to it.

If no size is specified, or a size of 0 bytes is set, then the parallel buffer becomes dynamic once again. Otherwise, size should be at least 5 bytes to ensure future compatibility.

**Example**

```
PAR_BUFF 10000
```

sets the parallel buffer to 10000 bytes.

**NOTE**

Although the SMSQ/E implementation allows a port to be specified, there are currently no implementations of the QL which have more than one parallel port, therefore trying to pass a port_number at present results in a bad parameter error.

**CROSS-REFERENCE**

*PRT_USE* sets up a dynamic printer buffer except under SMSQ/E.

## 23.19 PAR_CLEAR

| | |
|---|---|
| Syntax | PAR_CLEAR or<br>PAR_CLEAR port_number(SMSQ/E only) |
| Location | ST/QL, SMSQ/E |

This clears out all currently closed PAR buffers, thus preventing any further output. Any channels which are open to the PAR port will be left unaffected.

**NOTE**

Although the SMSQ/E implementation allows a port to be specified, there are currently no implementations of the QL which have more than one parallel port, therefore trying to pass a port_number at present results in a bad parameter error.

**CROSS-REFERENCE**

*SER_CLEAR* and *PRT_CLEAR* are similar.

## 23.20 PAR_DEFAULTPRINTER$

| | |
|---|---|
| Syntax | name$ = PAR_DEFAULTPRINTER$ |
| Location | SMSQ/E for QPC |

This returns the name of Windows' default printer. The name can later be used with *PAR_SETPRINTER* for example.

## 23.21 PAR_GETFILTER

| | |
|---|---|
| Syntax | state% = PAR_GETFILTER(port%) |
| Location | SMSQ/E for QPC |

This returns whether the printer filter is enabled for the specified port.

## 23.22 PAR_GETPRINTER$

| | |
|---|---|
| Syntax | name$ = PAR_GETPRINTER$(port%) |
| Location | SMSQ/E for QPC |

This returns the PAR port setting: "LPT1", "LPT2" or "LPT3" if it isn't linked to a printer but directly to a printer port or the name of the printer otherwise. An empty string designates the default printer.

## 23.23 PAR_PRINTERCOUNT

| | |
|---|---|
| Syntax | n% = PAR_PRINTERCOUNT |
| Location | SMSQ/E for QPC |

This returns the number of printers available on this system.

## 23.24 PAR_PRINTERNAME$

| | |
|---|---|
| Syntax | name$ = PAR_PRINTERNAME$(n) |
| Location | SMSQ/E for QPC |

This returns the name of printer number n (counted from 1 to *PAR_PRINTERCOUNT*).

## 23.25 PAR_PULSE

| | |
|---|---|
| Syntax | PAR_PULSE x |
| Location | ST/QL, SMSQ/E for the Atari |

Some accelerator boards enhance the speed of the Atari ST and TT computers so much that the parallel printer port may be affected. This can be fixed by using PAR_PULSE to increase the rate of the strobe pulse. This problem tends to be required if you have a printer which has heavy drive requirements (notably a CANON) or if you use a long printer cable.

**Example**

```
PAR_PULSE 80
```

**NOTE**

On SMSQ/E running on non-Atari's, this command has no effect.

## 23.26 PAR_SETFILTER

| | |
|---|---|
| Syntax | PAR_SETFILTER port%, state% |
| Location | SMSQ/E for QPC |

Enables (state% = 1) or disables (state% = 0) the printer filter for the specified port. If the printer should be enabled although none is available a "not found" error is returned.

## 23.27 PAR_SETPRINTER

| | |
|---|---|
| Syntax | PAR_SETPRINTER port%, name$ |
| Location | SMSQ/E for QPC |

Connects the PAR port either to a hardware port (**Example** name$ is "LPT1") or to the printer spooler (name$ is one of the names returned by *PAR_PRINTERNAME$*).

## 23.28 PAR_USE

| | |
|---|---|
| Syntax | PAR_USE [device] |
| Location | ST/QL, SMSQ/E, SuperQBoard, PAR/SER Interfaces, Super Gold Card |

As with many other devices, such as RAM, FLP and WIN, it can be useful to alter the three letter description used to access the parallel printer port on the Atari ST. The command PAR_USE allows you to replace the three letter description by any other three letters. If device is not specified, this changes the description back to PAR.

**Example**

```
PAR_USE ser
```

will divert any attempt to access the serial ports to the parallel printer port.

**CROSS-REFERENCE**

*RAM_USE*, *FLP_USE*, *WIN_USE*, *SER_USE* and *PRT_USE* are all very similar.

## 23.29 PAUSE

| | |
|---|---|
| Syntax | PAUSE [timeout] or<br>PAUSE [#chan,] [timeout](Minerva v1.80+, THORv6.41, SMS, ST/QL E-init v1.27+) |
| Location | QL ROM |

The command PAUSE halts execution of a program temporarily for the specified timeout number of frames (there are 50 frames per second in the UK and Europe, 60 frames per second in the US). If no timeout or a negative timeout is specified, the command will wait indefinitely. If a timeout of zero is specified, no actual PAUSE will take place. Execution will continue at the end of the timeout, or if a key is pressed. The key is read from channel #0 and therefore the command will report the error 'channel not open' if #0 is not open.

The second variant of this command allows you to specify a channel #chan (default #0) upon which the command should operate, thus allowing the command to be used in programs which do not have #0 open.

**Example**

```
PAUSE 100
```

Pauses for approximately 2 seconds, unless a key is pressed in the meantime!

**NOTE 1**

Using timeouts allows programs to run at the same speed on all QL implementations.

**NOTE 2**

Normally, if #0 or the specified channel (in the THOR variant of this command) is not a console window (or not open), an error will be generated, of either 'Bad Parameter' or 'Channel not open' respectively. However, the Minerva and SMS variants of this command do not report any error messages and merely return straight away (although see next note).

**NOTE 3**

On Minerva (v1.90+), the second variant of this command will also work on a screen (scr_) channel.

**NOTE 4**

The second variant didn't really work on ST/QL Emulators until v1.30 of E-Init).

**CROSS-REFERENCE**

*INKEY$* allows you to read the key which has been pressed, as well as halting program execution.

## 23.30 PE_BGOFF

| | |
|---|---|
| Syntax | PE_BGOFF |
| Location | SMSQ/E >= 3.12 |

Classic versions of the Pointer Environment suspend any jobs that are "buried" in the window stack and try to output onto their window. Some system extensions like PIE, PICE and PEX implemented workarounds for this, allowing jobs to continue running in the background even with output to the screen.

Starting from version 3.12, SMSQ/E supports background window I/O and update natively. This feature is enabled and disabled by the commands *PE_BGON* and PE_BGOFF.

**Example**

```
PE_BGOFF
```

Disables background window I/O.

**CROSS-REFERENCE**

See *PEOFF*, *PIE_ON*, *PXOFF*, *PE_BGON*.

## 23.31 PE_BGON

| | |
|---|---|
| Syntax | PE_BGOFF |
| Location | SMSQ/E >= 3.12 |

Classic versions of the Pointer Environment suspend any jobs that are "buried" in the window stack and try to output onto their window. Some system extensions like PIE, PICE and PEX implemented workarounds for this, allowing jobs to continue running in the background even with output to the screen.

Starting from version 3.12, SMSQ/E supports background window I/O and update natively. This feature is enabled and disabled by the commands PE_BGON and *PE_BGOFF*.

**Example**

```
PE_BGON
```

Enables background window I/O.

**CROSS-REFERENCE**

See *PEOFF*, *PIE_ON*, *PXOFF*, *PE_BGOFF*.

## 23.32 PEEK

See *PEEK_L* below.

## 23.33 PEEK_FLOAT

| | |
|---|---|
| Syntax | value = PEEK_FLOAT(address) |
| Location | DJToolkit 1.16 |

This function returns the floating point value represented by the 6 bytes stored at the given address. BEWARE, although this function cannot detect any errors, if the 6 bytes stored at 'address' are not a proper floating point value, the QL can crash. The crash is caused by QDOS and not by PEEK_FLOAT. This function should be used to retrieve values put there by *POKE_FLOAT* mentioned above.

**EXAMPLE**

```
1000 addr = RESERVE_HEAP(6)
1010 IF addr < 0 THEN
1020    PRINT "OUT OF MEMORY"
1030    STOP
1040 END IF
1050 POKE_FLOAT addr, PI
1060 myPI = PEEK_FLOAT(addr)
1070 IF myPI <> PI THEN
1080    PRINT "Something went horribly wrong!"
1090    PRINT "PI = " & PI & ", myPI = " & myPI
1100 END IF
```

**CROSS-REFERENCE**

*POKE_STRING*, *PEEK_STRING*, *POKE_FLOAT*.

## 23.34 PEEK_STRING

| | |
|---|---|
| Syntax | a$ = PEEK_STRING(address, length) |
| Location | DJToolkit 1.16 |

The characters in memory at the given address are returned to a$. The address may be odd or even as no word for the length is used, the length of the returned string is given by the length parameter.

**EXAMPLE** The following set of functions return the Toolkit 2 default devices:

```
1000 DEFine FuNction TK2_DATA$
1010   RETurn TK2_DEFAULT$(176)
1020 END DEFine TK2_DATA$
1030 :
1040 DEFine FuNction TK2_PROG$
1050   RETurn TK2_DEFAULT$(172)
1060 END DEFine TK2_PROG$
1070 :
1080 DEFine FuNction TK2_DEST$
1090   RETurn TK2_DEFAULT$(180)
1100 END DEFine TK2_DEST$
1110 :
1120 :
1200 DEFine FuNction TK2_DEFAULT$(offset)
1210   LOCal address
1220   IF offset <> 172 AND offset <> 176 AND offset <> 180 THEN
1230      PRINT "TK2_DEAFULT$: Invalid Offset: " & offset
1240      RETurn ''
1250   END IF
1260   address = PEEK_L (SYSTEM_VARIABLES + offset)
1270   IF address = 0 THEN
1280     RETurn ''
1290   ELSE
1300     REMark this is a pointer to the appropriate TK2 default
1310     RETurn PEEK_STRING(address+2, PEEK_W(address))
1320   END IF
1330 END DEFine TK2_DEFAULT$
```

**CROSS-REFERENCE**

*POKE_STRING*, *PEEK_FLOAT*, *POKE_FLOAT*.

## 23.35 PEEK_W

See *PEEK_L* below.

## 23.36 PEEK_L

| | |
|---|---|
| Syntax | PEEK (address) where address=0,1,2,3,… and<br>PEEK_W (address) where address=0,2,4,6,… and<br>PEEK_L (address) where address=0,2,4,6,… |
| Location | QL ROM |

These three functions are complementary to POKE, POKE_W and POKE_L, in that instead of setting a byte, word or longword in memory, these three functions return the value of the byte, word or longword stored at the given address.

**NOTE 1**

Due to the way in which values are stored in memory, it can be difficult to read negative values. However, although PEEK will return an unsigned byte in the range 0..255, PEEK_W will return a signed word in the range -32768…32767 and PEEK_L a signed longword.

**NOTE 2**

Do not try to PEEK_W or PEEK_L with an odd address (eg. 10001) as this will cause an error unless you are using Minerva (see below).

**MINERVA NOTES**

As with the POKE commands, the PEEK functions on Minerva (version 1.77 or later) are very much enhanced and different. Minerva allows you to use PEEK_W and PEEK_L on odd addresses, eg:

```
PRINT PEEK_W(131073)
```

Minerva has also added to the usefulness of the PEEK, PEEK_W and PEEK_L functions by allowing them to access system variables, Minerva's System Xtensions and SuperBASIC variables. You will need a good book on QDOS (eg. QDOS/SMS Reference Manual) to find out what the possible values are.

The syntax for these extra commands is:

**Look at SuperBASIC variables**

```
PEEK (\\SBvar)
PEEK_W (\\SBvar)
PEEK_L (\\SBvar)

PEEK (\SBvar\Offset)
PEEK_W (\SBvar\Offset)
PEEK_L (\SBvar\Offset)
```

**Look at System variables**

```
PEEK (!!Sysvar)
PEEK_W (!!Sysvar)
PEEK_L (!!Sysvar)

PEEK (!Sysvar!Offset)
PEEK_W (!Sysvar!Offset)
PEEK_L (!Sysvar!Offset)
```

**Look at System Xtensions**

```
sx_base=PEEK_L(VER$(-2) + 124)
PEEK (sx_base + offset)
```

**SMS NOTES**

SMS has altered the PEEK functions so that they are able to access System variables and SuperBASIC variables, using the same format as Minerva.

**CROSS-REFERENCE**

Please see in particular *POKE*, *POKE_W*, and *POKE_L*. *PEEK$* reads a string stored in memory and contains some examples of the new variants introduced on Minerva and SMS. *PEEK_F* and *PEEKS* are also worth a look.

## 23.37 PEEKS

See *PEEKS_L* below.

## 23.38 PEEKS_W

See *PEEKS_L* below.

## 23.39 PEEKS_L

| | |
|---|---|
| Syntax | PEEKS(address) where address=0,1,2,3,… and<br>PEEKS_W(address) where address=0,2,4,6,… and<br>PEEKS_L(address) where address=0,2,4,6,… |
| Location | SMSQ/E and ATARI_REXT v2.17+ |

These three functions are only of any use on the Atari series of computers. They are the same as the normal forms of PEEK, PEEK_W and PEEK_L, except that they operate in Supervisor Mode and can therefore be used to read data direct from the Atari's IO hardware. On all other implementations they are the same as PEEK, PEEK_W and PEEK_L.

**CROSS-REFERENCE**

See *PEEK*. *POKES* is the complementary command. See *PROT_MEM* also.

## 23.40 PEEK$

| | |
|---|---|
| Syntax | PEEK$ (start_address,bytes) or<br>PEEK$ (start_address [,bytes])(BTool only) |
| Location | ATARI_REXT, SMS, TinyToolkit, BTool |

This function will read a specified number of bytes from start_address in memory onwards and return the result as a string.

For BTool the second parameter is optional. If bytes is not specified then BTool's PEEK$ will try to read a string in QDOS format (ie. two bytes specifying the length of the string followed by the string itself) from the location start_address, just like CVS$ does. This string will then be returned. Note that start_address must always be even if bytes is omitted.

**Example**

Do you know how many keywords, filenames, variables etc. are currently known to the interpreter? This program lists and counts them.

```
100 adr=BASICP(32): num=0
110 REPeat all_names
120   length=PEEK(adr)
130   IF NOT length THEN EXIT all_names
140   name$=PEEK$(adr+1,length)
150   PRINT name$,
160   adr=adr+length+1: num=num+1
170 END REPeat all_names
180 PRINT\\num!"names"
```

**SMS NOTE**

PEEK$ allows you to access the System Variables and SuperBASIC variables in the same way as PEEK (etc.). For example, the above short program may be re-written as:

```
100 adr=0: num=0
110 REPeat all_names
120   length=PEEK(\$20\adr)
130   IF NOT length THEN EXIT all_names
140   name$=PEEK$(\$20\\adr+1,length)
150   PRINT name$,:PAUSE 160 adr=adr+length+1: num=num+1
170 END REPeat all_names
180 PRINT\\num!"names"
```

**WARNING**

A string cannot be longer than 32766 characters and so an expression such as a$=PEEK$(0,40000) may lead to unpredictable effects. Be careful!

**CROSS-REFERENCE**

*POKE$* is the complementary procedure to *PEEK$*. *PEEK*, *PEEK_W* and *PEEK_L* read single bytes, words and long words from memory. *TTPEEK$* is the same as this function.

## 23.41 PEEK_F

| | |
|---|---|
| Syntax | PEEK_F (address) |
| Location | BTool |

PEEK_F is a function which reads six bytes from any position in memory, which it assumes is the internal representation of a SuperBASIC floating point number, and returns its value.

**WARNING**

PEEK_F will lead to a crash if the six bytes at address did not represent a valid floating point, compare this with CVF.

**CROSS-REFERENCE**

*POKE_F*, *CVF*, *MKF$* See also *PEEK$*

## 23.42 PEND

| | |
|---|---|
| Syntax | PEND (#channel) |
| Location | TinyToolkit |

PEND is a logical function and returns 1 if there is data waiting in the specified channel to be read and 0 if not.

**Example 1**

If the Window Manager is present, PEND can be used to check if a window is currently hidden, and therefore to decide whether information should be printed to that channel or not. Under the Pointer Environment, jobs which are trying to output data to a window channel cannot do so until the channel is activated (eg. by pressing <CTRL><C>).

The following program calculates a large sum and prints the current value of the calculation in a small window, however, the calculation itself will not stop if one switches to another window, thus hiding this one.

```
100 n=1546: sum=0
110 OPEN#3,"con_"&(6*LEN(n)+6)&"x12a0x0"
120 BORDER#3,1,3: INK#3,7: CLS#3
130 FOR i=1 TO n
140   sum=sum+i
150   IF PEND(#3) THEN PRINT#3;FILL$(" ",4-LEN(i));i
160 END FOR i
170 IF sum<>n*(n+1)/2 THEN BEEP 0,33,44,66,22,44
180 CLOSE#3
```

**Example 2**

Pipes should be used for communication between jobs. It is very bad practice to write information to a file and let the other job read it because other tasks may be affected.

Here are two programs which have to be compiled and executed to multitask. Both open a small window, the first job inputs text and then sends it to the second job which shows that text. Typing "end" will terminate both jobs.

The output job would work without PEND but would not be able to do something else whilst waiting for further input.

```
100 REMark Input Job
110 :
120 OPEN#3,con_50x30a30x40: PAPER#3,3
130 INK#3,7: BORDER#3,1,4: CLS#3
140 OPEN#4,pipe_communication_200
150 REPeat input_loop
160   INPUT#3,text$
170   PRINT#4,text$
180   IF text$=="END" THEN EXIT input_loop
190 END REPeat input_loop
200 CLOSE#3: CLOSE#4
```

```
100 REMark Output Job
110 :
120 OPEN#3,scr_50x30a100x40: PAPER#3,3
130 INK#3,7: BORDER#3,1,4: CLS#3
140 OPEN#4,pipe_communication
150 REPeat output_loop
160   IF PEND(#4) THEN
170     INPUT#4,text$
180     IF text$=="END" THEN EXIT output_loop
190     PRINT#3,text$
200   END IF
210 IF NOT RND(200): d$=DATE$: PRINT#3,d$(16 TO)
220 END REPeat output_loop
230 CLOSE#3: CLOSE#4
```

By the way, in this case it is not very efficient to separate the input and output jobs, but good terminal Emulators do this. You will also notice that the programs use named pipes which make it much easier for them to link up with each other. These named pipes are present in the latest version of the ST/QL Emulator as well as SMS. They are also provided by several public domain device drivers - See the appendix on devices for further details.

**NOTE**

PEND only works with channels which will accept input (not scr_) and reports an "end of file" error (ERNUM=-10, ERR_EF=1) if a connected output pipe has been closed.

Unfortunately, EOF cannot be used to trap the end of a named pipe early enough, so you have to ensure that the output pipe tells the accompanying input pipe that it is about to be closed.

**CROSS-REFERENCE**

See *TCONNECT* and *FILE_OPEN* about connecting two unnamed pipes. *EOF* checks if a file is at its end. *IO_PEND%* and NOT *EOFW* are identical to *PEND*.

## 23.43 PENDOWN

| | |
|---|---|
| Syntax | PENDOWN [#ch] |
| Location | QL ROM |

This command is part of the QL's turtle graphics set of commands, and places the pen to the down position in the specified window (default #1). When a window is first opened, the pen is set to the up position.

**CROSS-REFERENCE**

*PENUP* has the opposite effect to this command. Also see *MOVE*.

## 23.44 PENUP

| | |
|---|---|
| Syntax | PENUP [#ch] |
| Location | QL ROM |

This command places the turtle's pen to the up position in the specified window (default #1), thus preventing any further drawing.

**CROSS-REFERENCE**

See *PENDOWN* and *MOVE* for more details.

## 23.45 PEOFF

| | |
|---|---|
| Syntax | PEOFF [{ #ch \| chan_ID \| job_name$ }] |
| Location | PEX |

This command is similar to PIE_OFF except that it allows you to disable background screen access for specific multitasking jobs if you wish (reverting to the original Pointer Environment method of managing windows). The same parameters as for PEON can be used to specify the Jobs or windows to be affected.

**NOTE**

PEX should not be used with PIE.

**CROSS-REFERENCE**

Refer to *PEON*.

## 23.46 PEON

| | |
|---|---|
| Syntax | PEON [{ #ch \| chan_ID \| job_name$ }] |
| Location | PEX |

PEX is similar to the PIE system extension (see PIE_ON for more details), in that it allows buried programs to access the screen in the background. However, PEX cannot be used with PIE and is completely independent. PEX should be loaded after the Pointer Environment (notably the PTR_GEN file), and after

any other code which redefines the window handling of the QL (for example Lightning or Speedscreen). It must however be loaded before the History device (except on SMSQ/E which has a built in History device).

People who use PEX or PIE may like to also use another utility PICE which updates the QL screen at pre-defined time intervals so that any part of a window which is not buried will actually appear on screen (whether or not part of that window is buried). If you wish to use PICE, it is recommended to set the PICE job to a priority of 1 so as not to slow the system down too much.

The PEON command allows you to select which windows and Jobs should allow background screen access - this is important since the more programs which continue to run in the background, the slower your QL will appear!! If no parameter is specified, then background screen access is enabled for all Jobs.

You can however pass any number of parameters, which can be:

1. A SuperBASIC channel number for the current program;
2. A QDOS channel number (see CHANNELS) in which case PEX will only affect that specific channel;
3. The name of a Job (passed as a string - use a null string ("") for SuperBASIC). PEX will affect all windows used by that specified Job.

**NOTE 1**

PEX will not work on pre-JS ROMs. On JS and MG ROMs, its functionality is reduced in that it can only be used to allow a few machine code calls which do not access the screen to operate notwithstanding that the Pointer Environment would normally stop them from working from within a buried program. It is equivalent to:

```
FOR i=0 TO 127: x=IS_PTRAP(i,3)
```

**NOTE 2**

Some toolkits report errors when used alongside PEX and some Qliberated programs refuse to work.

**CROSS-REFERENCE**

See *PEOFF*, *PIE_ON*, *PXON*, *PEX_INI* and *IS_PEON* for more details. *IS_PTRAP* allows you to enable PEX for specific machine code routines.

## 23.47 PEX$

| | |
|---|---|
| Syntax | PEX$ |
| Location | PEX |

This function returns the date of assembly, version and sub-version of the PEX file.

**CROSS-REFERENCE**

*PEX_SAVE* alters the sub-version number. See also *QL_PEX* and *PIF$*.

## 23.48 PEX_INI

| | |
|---|---|
| Syntax | PEX_INI |
| Location | PEX |

This command initiates PEX and makes it take effect - it may be useful for example to set up the various programs and the windowing environment, using PEON and IS_PTRAP and then to start PEX working at that stage, by using this command.

**NOTE**

Some toolkits report errors when used alongside PEX and some Qliberated programs refuse to work.

**CROSS-REFERENCE**

See *PEON* for more general details. You should also see *PEX_XTD* and *PX1ST*.

## 23.49 PEX_SAVE

| | |
|---|---|
| Syntax | PEX_SAVE directory$ |
| Location | PEX |

This command stores the current settings of PEX in a file called PEX19_byt (for version 19.30) in the specified directory so that when you next load PEX (with LBYTES directory$&PEX_19_byt for example), you will not have to re-set all of the various settings. The sub-version number of PEX is increased by one.

**Example**

```
PEX_SAVE 'win1_start_'
```

will create the file win1_start_PEX19_byt.

**NOTE**

An underscore must appear at the end of directory$.

**CROSS-REFERENCE**

See *PEON* for more general details. The settings which are saved are set with the command *IS_PTRAP* and IS_PXLST. *PEX$* returns the sub-version number.

## 23.50 PEX_XTD

| | |
|---|---|
| Syntax | PEX_XTD |
| Location | PEX |

This command re-installs the keywords provided as part of PEX and can help overcome the problem of other toolkits re-defining PEX keywords.

**CROSS-REFERENCE**

See *PEON* for more general details. You should also see *PEX_INI* and *PX1ST*.

## 23.51 PHONEM

| | |
|---|---|
| Syntax | PHONEM (string$) |
| Location | Ähnlichkeiten |

This function returns a string (even though the name does not end with $) which is a (more Germanic) phonetic transcription of the supplied string. If two words sound similar or are even homophones, their PHONEM's are identical. The function is not case-sensitive.

**Examples**

```
A$ = PHONEM ("Toolkit")
A$ = PHONEM ("DoolGid")
A$ = PHONEM ("DOLGYD")
```

All of which return "DOLCYD".

**NOTE**

An expression such as:

```
PRINT PHONEM (a$) INSTR PHONEM (b$)
```

will always return 0 on pre Minerva ROMs. Use temporary variables to get around this problem:

```
t1$=PHONEM(a$) : t2$=PHONEM(b$)
PRINT t1$ INSTR t2$
```

which will work properly.

**CROSS-REFERENCE**

*SOUNDEX*, *WLD*.

## 23.52 PI

| | |
|---|---|
| Syntax | PI |
| Location | QL ROM |

This function is a constant number which returns the value of Pi with an error of 10^(-29). You can test the precision of PI with such a program:

```
100 p = PI - 3: PRINT "PI = 3.";
110 FOR n = 1 TO 35
120   p = p * 10
130   PRINT INT(p);
140   p = p - INT(p)
150 END FOR n
160 PRINT
```

**CROSS-REFERENCE**

The trigonometrical functions (*SIN*, *TAN*, *ACOS* etc) expect parameters in radians, so *PI* must be used in most cases. Another fundamental mathematical constant, e, can be obtained with *EXP*(1).

## 23.53 PICK$

| | |
|---|---|
| Syntax | PICK$ (n, slct$^{1}$$ *[,slct$^{i}$$]*) |
| Location | CONTROL (DIY Toolkit Vol E) |

The function PICK$ takes one positive integer n and one or more other parameters slct1$, slct2$, etc. The function returns the value of the nth parameter, so n must be smaller than or equal to the number of supplied slctx$. Don't forget, n must be greater than zero!

**Example**

PICK$ is intended to simplify expressions, here are some silly examples:

```
100 bool%=RND (0 TO 1)
110 IF bool% THEN PRINT "yes": ELSE PRINT "no"
```

becomes:

```
100 bool%=RND (0 TO 1)
110 PRINT PICK$ (bool%+1,"no","yes")
```

Whereas:

```
100 members = RND (4)
110 PRINT "The team has ";
120 IF members>0: PRINT members;: ELSE PRINT "no";
130 PRINT " member";
140 IF members<>1 THEN PRINT "s": ELSE PRINT
```

becomes:

```
100 members = RND(4)
110 PRINT "The team has ";
120 PRINT PICK$ (1+(members<>0),"no",members);
130 PRINT " member";PICK$ (1+(members<>1),"","s")
```

The slightly more complex:

```
100 DIM num$ (9,5)
110 RESTORE : FOR i=0 TO 9: READ num$(i)
120 DATA "zero","one","two","three","four"
130 DATA "five","six","seven","eight","nine","ten"
140 :
150 REPeat typing
160 key = CODE (INKEY$ (-1))-48
170 IF key<0 OR key>9 THEN EXIT typing
180 PRINT num$ (key)!!
190 END REPeat typing
```

becomes:

```
100 REPeat typing
110 key = CODE (INKEY$(-1))-48
120 IF key<0 OR key>9 THEN EXIT typing
130 PRINT PICK$ (key+1,"zero","one","two","three","four","five","six","seven",
↪"eight", "nine", "ten")
140 END REPeat typing
```

**CROSS-REFERENCE**

Note that conditions have a numeric value, see *IF*, *AND* and *OR* for details. *SELect ON*.. END SELect <KeywordsS.clean.html#-end-select>`__ is a less restrictive alternative to *PICK$*.

## 23.54 PICK%

| | |
|---|---|
| Syntax | PICK% [(JobID] or<br>PICK% (JobID, action) |
| Location | PEX |

This function can be used to perform various acts.

The first syntax is the more common and will bring the specified Job (by reference to its QDOS Job id or its Job Number as specified by JOBS) to the top of the pile under the Pointer Environment, making all of its windows appear on screen as if it had been Picked from the Qpac 2 file menu.

If JobID is -1 or omitted, then the Job which contains this command, ie the current job, is brought to the top of the pile. This variant is therefore similar to TOP_WINDOW.

If JobID is -2, then the next Job in the Job Table (see NXJOB) is brought to the top of the pile - this is therefore similar to pressing <CTRL><C>.

The second variant is more complex and depends upon the values of JobID and action.

1. If JobID refers to an existing Job and action is -4, then the screen is frozen - this is equivalent to pressing <CTRL><F5>.
2. If JobID equals -3 and action is an existing QDOS channel number (see CHANNELS) or SuperBASIC channel number then that particular channel is unfrozen, allowing input from / output to that channel provided that the Job which owns that channel is at the top of the pile or can use background screen access under PEX.
3. If JobID equals -4 and action is an existing QDOS channel number (see CHANNELS) or SuperBASIC channel number then that particular channel is frozen again and any attempt by a program to access that channel will (unless that program is not buried) cause that program to halt temporarily.

The values returned by PICK% are normally zero if the function is successful. Otherwise errors are returned as follows:

- -1 : Job is In Use (although we are not certain when this will be reported).
- -2 : Job does not exist.
- -6 : Specified QDOS channel number does not exist.

**CROSS-REFERENCE**

*OJOB* and *NXJOB* allow you to find out details about a specified Job. *PEON* allows background screen access.

## 23.55 PIE_EX_OFF

| | |
|---|---|
| Syntax | PIE_EX_OFF |
| Location | PIE |

PIE_ON contains details about what PIE is used for and you should first of all refer to that.

Presuming that PIE is enabled (with PIE_ON), you may want to treat any programs (or toolkits) which use SD.EXTOP machine code calls to access the screen differently.

Normally, the Window Manager halts any program which attempts to call the SD.EXTOP machine code routine unless that program does not have any buried windows. However, PIE_ON allows all buried programs to continue in the background, storing the changes to the screen as necessary.

However, SD.EXTOP routines may be used for various purposes including writing to the screen directly and for this reason, if PIE is active, you may find that a program writes to the screen using SD.EXTOP even though its windows are buried (thus overwriting part of an existing program's display).

PIE_EX_OFF prevents this effect by still halting any program which attempts to use SD.EXTOP.

**CROSS-REFERENCE**

See *PIE_ON* and *PIE_EX_ON*. See also *PXOFF* which is similar.

## 23.56 PIE_EX_ON

| | |
|---|---|
| Syntax | PIE_EX_ON |
| Location | PIE |

This command re-enables PIE for SD.EXTOP system calls, so that they are affected by the normal PIE_ON and PIE_OFF commands.

**CROSS-REFERENCE**

See *PIE_ON*.

## 23.57 PIE_OFF

| | |
|---|---|
| Syntax | PIE_OFF |
| Location | PIE |

See PIE_ON below.

## 23.58 PIE_ON

| | |
|---|---|
| Syntax | PIE_ON |
| Location | PIE |

The Window Manager forms part of the Pointer Environment and is a standard system extension. It allows you to multitask all kinds of programs easily, provides non-destructible windows and more.

One of the main problems with current versions of the Window Manager is that if any part of the windows owned by a given Job is buried under another Job's windows (ie. you cannot see that part of the window on-screen because of another program), then if that buried Job tries to access the screen (with PRINT for example), the Window Manager will pause that Job until its window is no longer buried.

The Pointer Interface Extension (PIE) modifies the Pointer Environment so that a program is not halted when it tries to send screen output while its window is fully or partially buried by another job.

It does this by storing the changes to the buried window in memory and then when the buried Job is brought to the top of the pile (see PICK%), then its window is updated.

PIE_ON enables PIE, PIE_OFF disables it. These commands on their own cannot lead to any problems, you can switch PIE on and off whenever you like.

**CROSS-REFERENCE**

*PIE_EX_ON* and *PIE_EX_OFF*. See also *PEON* and *PXON* which greatly enhance these facilities. *PEND* can be used to check if a Job can send output to the screen.

## 23.59 PIF$

| | |
|---|---|
| Syntax | PIF$ |
| Location | PEX |

This is the same as QRAM$!

## 23.60 PINF$

| | |
|---|---|
| Syntax | PINF$ |
| Location | Fn |

This is the same as QRAM$ and PIF$!

## 23.61 PIXEL%

| | |
|---|---|
| Syntax | PIXEL% ( [#ch,] x1,y1 ) |
| Location | PIXEL (DIY Toolkit - Vol G) |

This function can be used to read the colour of a point in absolute co-ordinates on the screen with reference to the specified window channel (if any - default #1).

This function will work in MODE 4, 8 and 12 (on the THOR XVI, if you have v0.9+). The main limitation on this function is that the point must appear within the specified window, so x1 and y1 cannot exceed the width or height of the specified window (in pixels), or be less than zero.

**NOTE**

Although PIXEL% will work wherever the screen base is located, prior to v1.0, it assumed that a line of pixels takes 128 bytes - early versions will not therefore work on higher resolutions.

**CROSS-REFERENCE**

*PLOT* and *DRAW* allow you to draw points and lines on the screen. *INK* gives details about the various colour values which may be returned (this will be in the range 0…16).

## 23.62 PJOB

| | |
|---|---|
| Syntax | PJOB (job_ID) or<br>PJOB (jobnr,tag) or<br>PJOB (jobname) |
| Location | Toolkit II |

Each job has a priority - the function PJOB finds it and returns 0 if the given job does not exist, otherwise it returns the priority of the specified job.

You can calculate the job_ID with the formula:

job_id = jobnr + tag * $2^{16}$

A negative job_ID (preferably -1) identifies the job calling PJOB. The higher the priority, the more working time a job draws from the processor, and therefore the faster the execution.

**Example**

The priority of the main SuperBASIC interpreter can be seen with:

```
PRINT PJOB(0)
```

**MINERVA NOTE**

The maximum priority for a job on a standard QL is 255, however, on a Minerva ROM, the acceptable priority range is altered to -128…127. If PJOB returns a value over 127, then deduct the difference between this and 256 from zero to get the priority on a Minerva machine - see SPJOB for further details.

**CROSS-REFERENCE**

*JOB$*, *OJOB* and *NXJOB* return other information about a job.

## 23.63 PLAY

| | |
|---|---|
| Syntax | PLAY nr, music$ |
| Location | ST/QL, QSound |

The command PLAY will store the sequence music$ under the sequence number nr. The sequences are numbered 1, 2, 3, etc. No details are available for the limits of nr and the structure of music$.

**CROSS-REFERENCE**

*RELEASE* nr plays a sequence. *SND_EXT*.

## 23.64 PLOT

| | |
|---|---|
| Syntax | PLOT x,y,colour |
| Location | Fast PLOT/DRAW Toolkit |

This command forces a pixel to be set at the absolute screen position x,y. The origin (0,0) is the upper left corner of the full QL screen, the opposite corner (diagonally) is (511,255). Two neighbouring points do not have any space between them.

Co-ordinates greater than 511 (x) or 255 (y) or smaller than 0 are modulated - (x MOD 511) and/or (y MOD 255). The base address of the screen used by PLOT is defined with SCRBASE. PLOT works in MODE 4 only.

**Example**

The following procedure plots a point given in polar co-ordinates. A simple approach to draw a line in a polar system is listed at DRAW. A sensible range for the radius is 0 <= r <= 127.

```
100 DEFine PROCedure POLAR_PLOT (r,phi,col)
110   LOCal x,y
120   x=1.35*r*SIN(phi+PI/2)+255
130   y=r*COS(phi+PI/2)+127
140   PLOT x,y,col
150 END DEFine POLAR_PLOT
```

**NOTE 1**

PLOT writes directly into screen memory and will work on 512x256 resolutions only, it assumes by default that the screen starts at $20000 (redefine with SCRBASE) and works in MODE 4 only.

**NOTE 2**

Minerva users can SCRBASE SCREEN(#3) to allow PLOT to draw on the screen on which channel #3 is located.

**CROSS-REFERENCE**

*DRAW* draws a line, *SCLR* clears the screen, and *REFRESH* makes the screen defined by *SCRBASE* visible. Compare the other implementation of *PLOT*.

---

## 23.65 PLOT

| | |
|---|---|
| Syntax | PLOT [#ch,] x1,y1 |
| Location | DRAW (DIY Toolkit - Vol G) |

This command plots a point in absolute co-ordinates on the screen with reference to the specified window channel (if any - default #1). This is also used to specify the start point of a line to be drawn with the DRAW command from the same toolkit.

This is quicker than using the SuperBASIC POINT command and unlike other similar commands, it will support the current INK colour and OVER mode.

<CTRL><F5> will pause the point drawing and it will even work in MODE 4, 8 and 12 (on the THOR XVI, if you have v1.6+).

The main limitation on this command is that the point must appear within the specified window, so x1 and y1 cannot exceed the width or height of the specified window (in pixels), or be less than zero.

**NOTE**

Although PLOT will work wherever the screen base is located, it assumes that a line of pixels takes 128 bytes - it will not therefore work on higher resolutions.

**CROSS-REFERENCE**

See the other variant of *PLOT*. See also *DRAW*. Compare *POINT*.

## 23.66 POINT

| | |
|---|---|
| Syntax | POINT [#ch,] x,y *[;x$^i$,y$^i$]* |
| Location | QL ROM, GPOINT |

This command draws one or more specified points in the given window (default #1). The co-ordinates are floating point numbers, which means that two POINTs drawn with:

```
POINT 20,20: POINT 21,20
```

or:

```
POINT 20,20; 21,20
```

for example, are not normally neighbours. If a point lies outside a window, it is simply not drawn, ie. overflow errors do not occur. The graphics cursor is updated to the last point to be plotted.

**Examples**

```
POINT 50,50
POINT 50,50; 60,60
POINT #2,20,50
POINT #2,20,50; 50,20; 20,20; 50,50
```

**NOTE**

On MGx ROMs, there is a well-known POINT bug which makes POINT draw two points instead of one. This is fixed by some versions of Toolkit II, the ST/QL Emulator, SMS, Gold Card, other ROMs (especially Minerva) and small patches like GPOINT. GPOINT includes two commands: a replacement POINT and GPOINT which is just another name for the same thing.

**CROSS-REFERENCE**

The relation between the supplied co-ordinates and their position in the window is defined with *SCALE*. The colour of the point(s) is set with *INK*. The window can be resized with *WINDOW*. *LINE* draws a line. The *GPOINT* command is fully identical to *POINT* except that it fixes the MGx ROM bug. Check the ROM version with *VER$*.

## 23.67 POINT_R

| | |
|---|---|
| Syntax | POINT_R [#ch,] x,y *[;x$^i$,y$^i$]* |
| Location | QL ROM |

This command is similar to POINT except that all co-ordinates given are relative to the current graphics pointer.

**CROSS-REFERENCE**

See *POINT*! Also see *LINE_R* and *CIRCLE_R*.

## 23.68 POKE

See *POKE_L* below.

## 23.69 POKE_FLOAT

| | |
|---|---|
| Syntax | POKE_FLOAT address, value |
| Location | DJToolkit 1.16 |

This procedure will poke the 6 bytes that the QL uses to represent a floating point variable into memory at the given address. The address can be odd or even as the procedure can cope either way.

**EXAMPLE**

```
1000 Address = RESERVE_HEAP(6)
1010 IF Address < 0 THEN
1020    PRINT "ERROR " & Address & " Allocating heap space."
1030    STOP
1040 END IF
1050 POKE_FLOAT Address, 666.616
```

**CROSS-REFERENCE**

*POKE_STRING*, *PEEK_STRING*, *PEEK_FLOAT*.

## 23.70 POKE_STRING

| | |
|---|---|
| Syntax | POKE_STRING address, string |
| Location | DJToolkit 1.16 |

This procedure simply stores the strings contents at the given address. Only the contents of the string are stored, the 2 bytes defining the length are not stored. The address may be odd or even.

If the second parameter given is a numeric one or simply a number, beware, QDOS will convert it to the format that would be seen if the number was *PRINT*ed before storing it at the address. For example, 1 million would be '1E6' which is arithmetically the same, but characterwise, very different.

**EXAMPLE**

```
1000 Address = RESERVE_HEAP(60)
1010 IF Address < 0 THEN
1020    PRINT "ERROR " & Address & " Allocating heap space."
1030    STOP
1040 END IF
1050 POKE_STRING Address, "DJToolkit " & DJTK_VERS$
```

**CROSS-REFERENCE**

*PEEK_STRING*, *PEEK_FLOAT*, *POKE_FLOAT*.

## 23.71 POKE_W

See *POKE_L* below.

## 23.72 POKE_L

| | |
|---|---|
| Syntax | POKE address,value or<br>POKE address, value$^1$ *[,value$^i$]* (Minerva and SMS only) or<br>POKE address, {value$^1$ \| value$^1$$} *{,value$^i$ \| value$^i$$}* (SMS only)<br>and<br>POKE_W address,value or<br>POKE_W address, value$^1$ *[,value$^i$]* (Minerva and SMS only)<br>and<br>POKE_L address,value or<br>POKE_L address, value$^1$ *[,value$^i$]* (Minerva and SMS only) |
| Location | QL ROM |

Both kinds of internal memory (RAM and ROM) are organised as a stream of values. The basic unit for memory is a bit (a value of 0 or 1 relating to false or true), which relates to the binary system of counting.

Eight bits are combined to form a byte (0 to 255), sixteen bits make a word, and thirty-two a longword. Words and longwords are signed whilst bytes are unsigned.

The POKE commands allow you to set values in memory.

It is however unwise to POKE just anywhere, because there could be important code present in that part of memory which will be disrupted by POKEs and could crash your computer. You would generally have already set aside a part of memory for use by your own programs, by using RESPR or ALCHP and then you would POKE different values in that part of memory, eg. for storing data. This is a representation of the relationship between bits, bytes, words and longwords:

| | | | | |
|---|---|---|---|---|
| Bits | 10011000 | 10001000 | 11011111 | 10111000 |
| Bytes | 152 | 136 | 223 | 184 |
| Words | -26488 | | -8264 | |
| Long Word | -1.73586E9 | | | |

or

| | | | | |
|---|---|---|---|---|
| Bits | 01110110 | 11000111 | 01100000 | 00000011 |
| Bytes | 118 | 199 | 96 | 3 |
| Words | 30407 | | 24579 | |
| Long Word | 1.992778E9 | | | |

**NOTE 1**

Negative values can also be stored in memory. However, they are stored by deducting the number from the maximum number which can be stored in a byte plus one.

```
POKE 131072,255
POKE 131072,-1
```

have the same effect.

**NOTE 2**

Do not try to POKE_W or POKE_L to an odd address (eg. 10001) as this will cause an error unless you are using Minerva (see below).

**NOTE 3**

If you try to poke a value which is too big to fit into the given space, eg:

```
POKE 131072, -32768
```

then only the least significant byte is used (with POKE) and the low word is used (with POKE_W).

**NOTE 4**

The THOR XVI limits value to the following ranges: POKE: -128..255; POKE_W: -32768..65535

**MINERVA NOTES**

The POKE, POKE_W and POKE_L commands on Minerva (version 1.77 or later) are very much enhanced and different. Minerva allows you to POKE_W and POKE_L to odd addresses. eg:

```
POKE_W 131073,100100
```

Minerva has also added to the usefulness of the POKE, POKE_W and POKE_L commands by allowing them to store a list of numbers in one go.

As an example the following two program lines have exactly the same effect, although only line 100 will run on a non-Minerva QL.

```
100 POKE_W start,10: POKE_W start+2,125: POKE_W start+4,10322
110 POKE_W start,10,125,10322
```

Minerva also allows the BASIC programmer to access the QL's SuperBASIC variables, system variables and Minerva's own System Xtensions (although the extended PEEKs should be of more use). You will need a good book on QDOS (eg. QDOS/SMS Reference Manual) to find out what the possible values are. The syntax for these extra commands is:

**Alter SuperBASIC variables**

```
POKE \\SBvar,value: REMark SBvar=0...256
POKE_W \\SBvar,value
POKE_L \\SBvar,value
```

```
POKE \SBvar\Offset,value
POKE_W \SBvar\Offset,value
POKE_L \SBvar\Offset,value
```

**Alter System variables**

```
POKE !!Sysvar,value: REMark Sysvar=0...1152
POKE_W !!Sysvar,value
POKE_L !!Sysvar,value
```

```
POKE !Sysvar!Offset,value
POKE_W !Sysvar!Offset,value
POKE_L !Sysvar!Offset,value
```

The command

```
POKE \\SBvar,value
```

will alter the SuperBASIC variable pointed to by Sysvar, such as the current line number. The most useful of these are variables $68 onwards.

The command

```
POKE \SBvar\Offset,value
```

allows you to alter the different SuperBASIC tables used by the QL (eg. the channel table). The start addresses of the different tables are contained in the SuperBASIC variables $0 to $64. SBvar must contain the relevant SuperBASIC variable (the pointer to the required table), then the Offset is the required address within the table.

The command

```
POKE !!Sysvar,value
```

allows you to alter the different system variables (normally stored at $28000 on a QL, but they can move!). These are useful for accessing the current network number, finding free space, accessing device drivers, forcing <CAPSLOCK>..... Sysvar merely contains the number of the required system variable.

The command

```
POKE !Sysvar!Offset,value
```

takes the address contained within the given system variable, adds the Offset to that address and then pokes it with the given value.

On a Minerva machine the system variable stored at $7C (124) (SV.CHTOP) contains the address of the Minerva System Xtensions, therefore to alter these:

```
SysX = PEEK_L (ver$(-2) + 124)
POKE SysX + offset,value
POKE_W SysX + offset,value
POKE_L SysX + offset,value
```

Minerva's System Xtensions provide such things as the addresses for translation tables, the attributes for the size type and colour of a cursor, the fonts for all subsequently opened windows and much more… (see Minerva manual for list).

**Minerva Example 1**

It is sometimes useful to alter the key repeat delay and frequency to make allowances for when a different keyboard is attached to the QL, so that you can type more quickly without having the problem that you are waiting around for auto-repeat to take effect. These two values can now simply be altered using:

```
POKE_W !!140, key_delay
POKE_W !!142, key_frequency
```

**Minerva Example 2**

Want to attach a new font to all channels which will be opened in the future?

```
100 a=RESPR(2000)
110 LBYTES flp1_new_font, a
120 POKE_L !124!40, a
```

**Minerva Example 3**

It might be useful in an error trapping routine to find the current DATA position (eg. if there is an error when reading data into a variable), so that the position may be returned to later once the error has been overcome. You may even wish to miss out the problem DATA line. This program is an 'intelligent' data-loader:

```
100 WHEN ERRor
110 data_line=PEEK_W(\\148)
115 PRINT 'ERROR IN DATA LINE'!data_line!';statement'! PEEK(\\151)-1
120 INPUT 'Go to next data line (y/n)'!a$
130 IF a$=='y': POKE_W\\148,data_line+1: POKE\\150,1:POKE\\151,1: RETRY
140 IF a$=='n' THEN
145   data_store=PEEK_W(\\148)*65536+(PEEK(\\150)-1)*256+PEEK(\\151)-1
147   PRINT"Alter offending line then enter re_run":STOP
149 END IF
150 END WHEN 155 :
160 RESTORE
170 ax=RESPR(100):i=0
180 REPeat data_load
190   IF EOF: EXIT data_load
200   READ b
210   PRINT b,i:POKE ax+i,b
220   i=i+1
230 END REPeat data_load
240 DATA 10,10,2,3,3a,10
250 DATA 10,2,2,3,3,2
```

(continues on next page)

(continued from previous page)

```
255 :
260 DEFine PROCedure RE_run
270   POKE_L \\148,data_store: GO TO 170
280 END DEFine
```

**SMS NOTE**

POKE, POKE_W and POKE_L have been made the same as on Minerva except that POKE_W and POKE_L cannot address odd addresses.

SMS does not possess Minerva's System Xtensions.

Please also note that SMS's improved interpreter won't allow you to enter line 240 in the Minerva Example 3 as the data item 3a will be rejected.

One extra addition to SMS is that the POKE command can actually accept a string as a value to be poked into memory. If a string is passed as a parameter, each character of the string is converted to its character code and then that byte poked into memory, for example:

```
POKE base,0,5,'WIN1_'
```

will store 'WIN1_' as a standard QL string (a word containing its length followed by the string itself) at the address in memory pointed to by base. Note that if you pass an empty string, this will have no effect.

**WARNING**

**If you are POKEing around in memory then make sure that you know what you are doing.**

On every QDOS machine, even RAM areas which have not been set aside for program use are used by the operating system, eg. for buffering purposes. On Emulators and QLs fitted with a Gold Card, the operating system itself is no longer in ROM but is moved into RAM. POKEing in this area will almost surely lead to crashes. Even advanced users who know which parts of memory are used by QDOS should avoid amending QDOS directly. There are more elegant and safer methods how to do this which will run on every QDOS compatible computer.

**CROSS-REFERENCE**

*PEEK*, *PEEK_W*, *PEEK_L* and *PEEK$* read memory values and *POKE$* is another command to set them. *CHAR_DEF* allows you to attach a font to all channels *OPEN*ed after the command. *POKES* allows you to *POKE* memory in Supervisor mode.

## 23.73 POKES

See *POKES_L* below.

## 23.74 POKES_W

See *POKES_L* below.

## 23.75 POKES_L

| | |
|---|---|
| Syntax | POKES address, {value[1] \| value[1]$} *{,value[i] \| ,value[i]$}* <br> and <br> POKES_W address, value[1] *[,value[i]]* <br> and <br> POKES_L address, value[1] *[,value[i]]* |
| Location | SMSQ/E and ATARI_REXT v2.17+ |

These three commands are only of any use on the Atari series of computers. They are the same as the simple forms of POKE, POKE_W and POKE_L, except that they operate in Supervisor Mode and can therefore be used to write data direct into the Atari's IO hardware. On all other implementations they are the same as POKE, POKE_W and POKE_L.

**CROSS-REFERENCE**

See *POKE* and *PEEKS*. Also see *PROT_MEM*.

## 23.76 POKE$

| | |
|---|---|
| Syntax | POKE$ address,string$ |
| Location | ATARI_REXT, TinyToolkit, BTool, SMS |

The standard version of this command pokes the code of each of the given string's characters to memory from address onwards. In SuperBASIC, the procedure might look similar to:

```
100 DEFine PROCedure POKE$ (address,string$)
110   LOCal i
120   FOR i=1 TO LEN(string$)
130   POKE address+i-1,CODE(string$(i))
140   END FOR i
150 END DEFine POKE$
```

The BTool version writes the string in QDOS internal format: the string's contents are preceded by two additional bytes (one word) indicating the length of the string. address must be even. If you pass an empty string, all versions of this command will not do anything.

**SMS NOTE**

This command is now very similar to POKE in that POKE allows you to pass a string as a parameter. POKE$ can also now access the System Variables and SuperBASIC variables directly as with POKE.

**CROSS-REFERENCE**

*PEEK$* reads strings from memory. *MKS$* returns the internal format of a given string. *TTPOKE$* is the same as this command.

## 23.77 POKE_F

| | |
|---|---|
| Syntax | POKE_F address,float |
| Location | BTool |

Floating point numbers are internally stored as six bytes. POKE_F will store any float at address in memory where ODD(address)=0.

**Example**

Floating point numbers can be stored in internal format in a file with PUT. The disadvantage of that method is low disk access speed if you need to store a large number of values.

Compare the following two programs which store the same amount of data at different speeds.

Slow but minimal memory usage:

```
100 n=1000: file$="flp1_test_dat"
120 fp=FOP_NEW(file$)
130 FOR i=1 TO n: PUT#fp,RND
140 CLOSE#fp
```

Fast but 6K buffer required:

```
100 n=1000: file$="flp1_test_dat"
120 a=ALCHP(6*(n+1))
130 FOR i=0 TO n: POKE_F a+i*6,RND
140 SBYTES file$,a,6*(n+1)
150 RECHP a
```

**CROSS-REFERENCE**

*POKE*, *POKE_W* and *POKE_L* store different ranges of integer numbers. *MKF$* returns the internal representation of a floating point number as a string. *GET* and *PUT* write all kinds of data types in their internal format to a channel, *FPUTF* and *FGETF* are specialised variants for floats only. See also *PEEK_F*!

## 23.78 PRINT

| | |
|---|---|
| Syntax | PRINT [#chan,] *[ [separator] [strg$ separator] var]* or<br>PRINT *[ [#chan,] [separator] [strg$ separator] var]* (THOR XVI & Minerva v1.97+ only) |
| Location | QL ROM |

This command will send a string of bytes to the specified channel (default #1).

If any variables (var) are specified, the contents of those variables are PRINTed in the specified channel.

If the channel is a window, the characters printed appear at the current text cursor position, in the current INK colour on the current STRIP colour, and will also be affected by the settings of CSIZE, UNDER, FLASH and OVER.

If you tell PRINT to use an unset variable, an asterisk ('*') will be PRINTed on screen rather than an error being reported (except on SMS where an unset variable is given the value 0 (if a numeric variable) or '' for a string). Beware, however that if you try to use an unset variable in a calculation inside the PRINT statement, an 'Error in Expression' error will be generated, for example:

```
a=10 : PRINT 'A is :'! a ,'B is :'! b : PRINT 'A*B is :' !a*b
```

If a channel is specified, this must be followed by a comma. It may however also be followed by one or more separators, as with INPUT.

At the end of the PRINT command, the text cursor is placed at the start of the next print line (unless an end separator of '!', '\' or ';' is used). When using a separator of '!', TO or ',' on a non-window channel, the PRINT statement will always assume the end of each line to be the number of characters specified with the WIDTH command, thus allowing you to format your output on a printer, for example.

**Example**

The following procedure allows you to print text to a given channel without splitting words when the text wraps onto the next line:

```
100 DEFine PROCedure PRINT_TEXT(ch,txt$)
110   LOCal print_loop
120   REPeat print_loop%
130     IF LEN(txt$)=0:PRINT #ch:RETurn
140     I%=' ' INSTR txt$
150     IF I%=0:PRINT #ch!!txt$:RETurn
160     PRINT #ch!!txt$(1 TO I%-1)!
170     IF I%<LEN(txt$):txt$=txt$(I%+1 TO):ELSE txt$=''
180   END REPeat print_loop%
190 END DEFine
```

Try:

```
WINDOW #1,50,100,32,16: PRINT_TEXT #1,'This is a test line'
```

Compare:

```
PRINT #1,'This is a test line'
```

**NOTE 1**

Version 6.40 of the THOR XVI ROM can crash if you try to use PRINT with the '!' separator in a non-window channel.

**NOTE 2**

The THOR XVI (all versions) and non-Minerva ROMs (unless SMS is installed) have problems with the concatenation of values which should produce an 'Overflow Error'. For example:

```
PRINT 'hello'&(1/0)
```

may crash the computer rather than producing an overflow error.

**NOTE 3**

PRINT can only show a maximum of six *integer* digits. If a number is larger than this, it will be represented by the E function (eg. 1E2 is the same as 100). If on the other hand, the figure is a *floating point*,

then the QL can cope with seven digits excluding the decimal point, eg. 123.4567. Any more digits will cause the number to be rounded up or down as appropriate.

**MINERVA NOTE**

v1.97+ allows different channels part way through statement as per THOR XVI.

**THOR XVI NOTE**

Version 6.41 of the THOR XVI allows you to put channel numbers part way through a statement, for example:

```
PRINT 'Name:'!name$ \#0; 'Address:' !address$
```

instead of:

```
PRINT 'Name '!name$ : PRINT #0;'Address:'!address$
```

**CROSS-REFERENCE**

See also *INPUT* which contains a description of the different types of separators. Please also see *WIDTH*. *TO* has other meanings - see *TO*. *VG_PRINT* allows you to print using scalable fonts on screen. *CHAR_USE* and *CHAR_DEF* allow you to alter the fonts used for printing characters on screen.

## 23.79 PRINT_USING

| | |
|---|---|
| Syntax | PRINT_USING [#ch,] format$, *[item$^{i}$]* |
| Location | Toolkit II |

This command allows you to send output to the specified channel (default #1) in a particular format. This for example, allows you to print neat columns of figures easily, all lined up on the decimal point.

The format$ is made up of a mixture of special characters, text and fields. Basically, PRINT_USING will print out format$ as normal, until one of these special characters is met. The special characters currently supported are: © (copyright) + - ( # * , . ! \ ‘ “ and $. These have the following effects:

| Character | Effect |
|---|---|
| © | This forces PRINT_USING to print out the next character in format$ even if it is a special character. If you want to print some text including one of the special characters, this must be used. |
| + | This is used to either prefix or postfix a decimal field. If present, then the sign of the decimal number is written out in this position. |
| - | This is used to either prefix or postfix a decimal field. The sign of the decimal number will only be written in this position if the number is negative. This and the closing bracket can be used to surround a decimal field, in which case if the number is negative, it will appear in brackets. |
| # | (Hash) This is used to mark a type of field (see below). |
| * | (Asterisk) This is also used to mark a type of field (see below). |
| \ | This will force a newline to take place. Unlike PRINT, PRINT_USING does not automatically carry out a newline after finishing its work. |
| “ and ‘ | Anything between either single or double quotation marks will be printed out without looking for special characters. |
| $ | This is used to signify the start of a currency field. Any characters between this sign and the next ‘#’ symbol are taken to be the name of the currency and are pushed right to line up with the actual amount. |

The fields in the format$ allow you to print text and/or figures in specific formats. Each item following format$ is then read and inserted in place of each field. If however, a numeric field is not long enough to hold the specified figure, then the field appears as just ‘#’ marks on screen. Text fields will just truncate the text supplied to fit. The fields which are recognised are:

| Field | Meaning |
|---|---|
| #### | If item is text, write it left justified and truncate to fit size of field if necessary. If item is a number, write the integer part of the number right justified (eg. PRINT_USING '###','Hello' will print Hel). |
| **** | This is the same as #### except that any unused part of the field to the left of the characters is filled with '*' characters. (eg. PRINT_USING '****',1.234 will print ***1). |
| ###.## | Print a fixed decimal point number right justified to a set number of decimal places. (eg. PRINT_USING '##.#',1.26 will print 1.3). |
| ***.** | The same as ###.## except that any unused part of the field is filled with '*' characters. |
| #,###.## | This is the same as ###.## except that a comma will be used to separate thousands. |
| *,***.** | This is similar to #,###.## except that any unused part of the field will be filled with '*'. |
| -#.###!!!! | This is used for an exponential field with the sign only being shown if the figure is negative. (eg. PRINT_USING '-#.##!!!!',3120 will print 3.12E+03). An exponential field must always begin with a sign followed by one # mark and a decimal point, and always end with four ! marks. |
| +#.###!!!! | This is the same as -#.###!!!! except that the sign of the number is always shown. |
| ###.>> | This is introduced by SMSQ/E v2.73+ and is the same as ###.## except that it is for fixed point decimal figures, scaled accordingly. This allows you, for example, to convert a calculation from pennies into pounds. (eg. PRINT_USING '###.>>',312.01 will print 3.12). You can add more > characters after the decimal point if you need to convert to three decimal places. |
| ***.>> | This is introduced by SMSQ/E v2.73+ and is the same as ###.>> except that any unused part of the field is filled with '*' characters. |

**Example**

A program which prints out a stocklist, which might be useful for a small business:

```
100 RESTORE
110 MODE 4
120 WINDOW 448,200,32,16:PAPER 0:INK 7:CLS
130 CSIZE 2,0:AT 1,10:UNDER 1:PRINT 'STOCK LIST'
140 CSIZE 1,0:AT 5,0
150 PRINT 'NO ITEM IND. PRICE TOTAL'
160 UNDER 0
170 total=0:Lines=6
180 REPeat loop
190   IF EOF:EXIT loop
200   READ equipment$,items,ind_price
210   price=ind_price*items
220   total=total+price:Lines=Lines+1
230   PRINT_USING '#,###. #############',items,equipment$
240   PRINT_USING ' $##.## $##,###.##\',ind_price,price
250   END REPeat loop
260 OVER 1:AT Lines-1,0:UNDER 1
270 PRINT FILL$(' ',45):UNDER 0
```

(continues on next page)

(continued from previous page)

```
280 IF INT(total)<>total:total=total*100
290 PRINT TO 23;'Total Stock £';CDEC$(total,9,2)
1000 DATA 'Minerva',110,40,'Minerva MKII',205,65.61
1010 DATA 'Hermes',100,25,'68008 CPU',1230,8.7
```

**NOTE 1**

On Toolkit II versions before v2.08, this command could cause problems if an empty string was passed to it.

**NOTE 2**

Some versions of the Toolkit II manual get the copyright symbol ('©') mixed up with the 'at' symbol ('@'). The latter has no special meaning.

**NOTE 3**

Because of variations in the way in which numbers are represented in different countries, either a comma or full stop is recognised as a decimal point by PRINT_USING. If a field only contains one comma or full stop, that is taken to be the decimal point, however, if more than one comma and/or full stop appears in the field, the last one is taken to be the decimal point, the others being assumed to be thousands separators. If you want numbers to be printed with thousands separators but no decimal point, use a comma or full stop as the last character of the field.

**CROSS-REFERENCE**

See also *FDEC$*, *IDEC$* and *CDEC$*.

## 23.80 PRIO

| | |
|---|---|
| Syntax | PRIO priority |
| Location | PRIO |

This command sets the priority of the current job to the given priority. Priority must range from 0 to 127.

**Example**

Multitasking jobs waiting for a keypress or anything else to be activated slow down the whole system although they are actually doing nothing. A job which is waiting (perhaps for a certain amount of time) for input could set its own priority to one, and then when it is able to continue, reset to a higher priority value.

**NOTE**

If a job has priority 0 it will not be able to run. Other tasks may however set that job's priority (eg. with SPJOB and allow it to continue).

**CROSS-REFERENCE**

*SPJOB*, *SP_JOB*, and *PJOB* also deal with job priorities.

```
SPJOB -1
```

is exactly the same as PRIO, priority, or *PRIORITISE*.

## 23.81 PRIORITISE

| | |
|---|---|
| Syntax | PRIORITISE [jobnr, jobtag,] priority |
| Location | TASKCMDS (DIY Toolkit Vol J) |

PRIORITISE is a command which takes either one or three parameters and sets the priority of the current job (if only one parameter is used) or the job specified by jobnr and jobtag to priority.

**Example**

```
PRIORITISE 127
```

gives the current job the maximum amount of processor time available when multitasking.

**CROSS-REFERENCE**

A one parameter variant of *PRIORITISE* is *PRIO*. Refer to *SPJOB* and connected keywords for more information on jobs and priorities. *JOBS* will give details of job numbers and job tags.

## 23.82 PRO

| | |
|---|---|
| Syntax | PRO |
| Location | Beuletools |

This function returns the codes needed to switch on the proportional font on an EPSON compatible printer:

```
PRINT PRO
```

is the same as:

```
PRINT CHR$(27) & "p" & CHR$(1)
```

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*, *DBL*, *ENL*, *SI*, *NRM*, *UNL*, *ALT*, *ESC*, *FF*, *LMAR*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 23.83 PROCESSOR

| | |
|---|---|
| Syntax | PROCESSOR |
| Location | SMSQ/E |

This function returns a value which can be used to find the type of Processor on which SuperBASIC is running (normally a member of the Motorola 680xx family). The values returned are:

| PROCESSOR | Chip Type |
|---|---|
| 0x | 68000 or 68008 |
| 1x | 68010 or an INTEL chip (QPC < 3.33) |
| 2x | 68020 (and QPC >= 3.33) |
| 3x | 68030 |
| 4x | 68040 |
| 6x | 68060 |

In the above 'x' is replaced by a value between 0 and 8 to indicate if a maths co-processor is installed).

You can also test if a maths co-processor is installed, by using:

```
coprocessor%=PEEK(!!$A1) && BIN('1111')
```

The following values may be returned:

| Coprocessor | Meaning |
|---|---|
| 0 | No FPU fitted. |
| 1 | An Internal MMU is fitted. |
| 2 | A 68851 MMU is fitted. |
| 4 | An internal FPU is fitted. |
| 8 | Either a 68881 or 68882 FPU is fitted. |

**NOTE**

The processor type was not stored before Level E-20 of the ST/QL Drivers.

**QPC Note**

QPC versions prior to 3.33 will return a value of 10 for the PROCESSOR function while those from 3.33 onwards will return 20.

**CROSS-REFERENCE**

See *MACHINE*, *QDOS$* and *VER$*

## 23.84 PROCedure

| | |
|---|---|
| Syntax | … PROCedure |
| Location | QL ROM |

This keyword forms part of the structure DEFine PROCedure. As such, it cannot be used on its own within a program - this will cause a 'bad line' error.

**CROSS-REFERENCE**

Please refer to the individual structure descriptions for more details.

## 23.85 PROGD$

| | |
|---|---|
| Syntax | PROGD$ |
| Location | Toolkit II |

This function returns the default program device as set by PROG_USE, see below.

**CROSS-REFERENCE**

*PROG_USE*, *DLIST*, *DATAD$*.

## 23.86 PROG_USE

| | |
|---|---|
| Syntax | PROG_USE default_device |
| Location | Toolkit II, THOR XVI |

The command PROG_USE and dependent commands behave in the same way as DATA_USE with a few differences.

The program device set with PROG_USE is used by the EX (EXEC), EW (EXEC_W) and (exceptionally SEXEC) commands as the default device. Whereas some commands which use the data device (eg. MERGE, LOAD) check the program device if they do not find a given file on the data device, the above commands which use the program device will not look at the data device should they fail on the program device.

**NOTE**

The default devices cannot exceed 32 characters (plus a final underscore) - any attempt to assign a longer string will result in the error 'Bad Parameter' (error -15).

**CROSS-REFERENCE**

*PROGD$* returns the program device setting. See *DATA_USE* for more information.

## 23.87 PROT_DATE

| | |
|---|---|
| Syntax | PROT_DATE flag |
| Location | SMS, Gold Card |

Many systems which can run SMS (including QXL, the Gold Card and Super Gold Card) include a battery backed clock (also known as a real-time clock). In this case, there are actually two clocks running:

One is run by the operating system (the QL internal clock) which is found on each QL implementation. The internal clock forgets the time if the computer is switched off and has to be set each time the machine is powered up.

The other clock is the battery backed clock which keeps the time even when the QL is switched off (until the battery runs flat) and this normally sets the Internal Clock each time the QL is reset (or switched on).

It may be necessary to adjust the QL's internal clock whilst the QL is being used, without wishing to disrupt the battery backed clock - some software alters the QL's internal clock when there is no need, the internal clock can also be affected by crashes during program development.

Some battery backed clocks may alter their time when the QL's internal clock is altered and therefore some form of protection is needed - you will normally need to enable the protection by using the command:

```
PROT_DATE 1
```

```
PROT_DATE 0
```

will disable the protection.

**NOTE 1**

This has no effect on the battery backed clock provided by Minerva MKII which has to be altered using the configuration program.

**NOTE 2**

If you reset the Gold Card or Super Gold Card to 128K, PROT_DATE 1 is executed.

**NOTE 3**

Serious crashes and some old games may disturb the battery backed clock even in protected mode.

**NOTE 4**

On some combinations of AURORA and Super Gold Card, if you use PROT_DATE 1, the QL's internal clock will run too quickly (see also the notes on DISP_SIZE).

**WARNING**

SMS, the Gold Card and Super Gold Card do not automatically protect the battery backed clock. It is therefore advisable to include the line:

```
PROT_DATE 1
```

in your boot program.

**CROSS-REFERENCE**

*SDATE* and *ADATE* alter the QL's internal clock.*DATE$* and *DAY$* can be used to read the time on the QL's internal clock.

## 23.88 PROT_MEM

| | |
|---|---|
| Syntax | PROT_MEM level |
| Location | SMSQ/E |

The command PROT_MEM can be used to set the level of memory protection which is afforded on Atari ST and TT computers, to try and stop the user from altering essential areas of the operating system by mistake. There are five levels of memory protection currently available:

| Level | Protection |
|---|---|
| 0 | Memory access faults are not reported. |
| 1 | Write memory access faults are trapped from all jobs except from Job 0. Read operations from a protected area read 0. |
| 2 | Read memory access faults are trapped from all jobs except Job 0. Any Write operations to a protected area are ignored. |
| 3 | Both Read and Write memory access faults are trapped from all jobs except Job 0. |
| 7 | Both Read and Write memory access faults are trapped from all jobs. |

The default level is 3.

We would recommend that Level 0 is avoided if at all possible. Memory access faults tend to occur when the user (or a program) tries to access memory which does not exist or can only be accessed in Supervisor Mode (the vector area, the TOS system variables and the IO hardware). However, under SMSQ/E, if there is an attempt to read an address which actually forms part of the legitimate QL vector area, this will not cause a fault. If a memory access fault is trapped, the Job which has caused the fault is paused and the program counter is placed on the stack (all registers are preserved). An advanced user may then use a debugger to examine the Job to find out what has caused the fault.

**NOTE**

Unfortunately, on other implementations, this command has no effect, and it is therefore still possible to overwrite the operating system on QL Emulators (non-Atari based), Gold and Super Gold Cards.

**CROSS-REFERENCE**

See *POKES* and *PEEKS*. *PROT_DATE* protects a battery backed clock.

## 23.89 PROUND

| | |
|---|---|
| Syntax | PROUND (p, x) |
| Location | TRIPRODRO |

PROUND is a function which rounds the given floating pointer number x to the precision of $10^p$. It looks at the next digit to decide whether to round upwards or downwards and ignores any others.

**Example**

Print ten random number with three digits after the decimal point:

```
100 RANDOMISE
110 FOR i = 1 TO 10
120 PRINT PROUND(-3, 10*RND)
130 END FOR i
```

**CROSS-REFERENCE**

*DROUND*.

## 23.90 PRT_ABORT

| | |
|---|---|
| Syntax | PRT_ABORT |
| Location | ST/QL, SMSQ/E |

This is the same as PAR_ABORT or SER_ABORT, depending on which device PRT is linked to.

**CROSS-REFERENCE**

See *SER_ABORT* and *PAR_ABORT*. *PRT_USE* allows you to specify which port prt emulates.

## 23.91 PRT_ABT

| | |
|---|---|
| Syntax | PRT_ABT |
| Location | Trump Card, Gold Card, QXL |

Because all output sent to the Trump Card, Gold Card and Super Gold Card's built in PRT device is buffered (except if you are running SMSQ/E which uses its own PRT device), it can be useful to stop the port from outputting any further data.

PRT_ABT will prevent any further output and clear the contents. The message ****** ABORTED ****** will then be sent to the port.

**CROSS-REFERENCE**

*PRT_USE* allows you to specify the type of output to be buffered. The ST/QL Emulator and SMSQ/E support a similar function with *PRT_ABORT*, *PAR_ABORT* and *SER_ABORT*.

## 23.92 PRT_BUFF

| | |
|---|---|
| Syntax | PRT_BUFF [size] |
| Location | ST/QL, SMSQ/E |

This is exactly the same as PAR_BUFF except that it creates buffered output on whichever port is attached to the PRT device.

**CROSS-REFERENCE**

See *PAR_BUFF*.

## 23.93 PRT_CLEAR

| | |
|---|---|
| Syntax | PRT_CLEAR |
| Location | ST/QL, SMSQ/E |

This clears out all currently closed PRT buffers, thus preventing any further output, in the same way as PAR_CLEAR.

**CROSS-REFERENCE**

*PAR_CLEAR* and *SER_CLEAR* are similar.

## 23.94 PRT_USE

| | |
|---|---|
| Syntax | PRT_USE device |
| Location | ST/QL, SMSQ/E |

The ST/QL Emulator and SMSQ/E allow you to set up the PRT device so that it mimics the SER, STX or PAR device. This means that programs can be written which merely send their output to the PRT device and it is then up to the user to set the port and options required by the device attached to either the serial or parallel port.

The command PRT_USE allows you to specify both the port and options to be associated with PRT.

It will ignore SER_USE and PAR_USE settings and therefore expects device to be in one of the following forms:

```
PAR<port><translate><convert><eof>
SER<port><parity><handshake><translate><convert><eof>
STX<port><parity><handshake><translate><convert><eof>
```

See the Appendix on drivers for further details.

**Example**

```
PRT_USE ser1etf
```

will cause all attempts to access the PRT device to be re-directed to serial port 1 with Even parity, translation enabled and a form feed being printed at the end of the file.

**CROSS-REFERENCE**

*RAM_USE*, *FLP_USE*, *WIN_USE*, *SER_USE* and *PAR_USE* are all very similar. See the other version of this command.

---

## 23.95 PRT_USE

| | |
|---|---|
| Syntax | PRT_USE usage,device |
| Location | Qjump RAMPRT, Trump Card, Gold Card, QXL Card |

Unlike the ST/QL Emulator and SMSQ/E implementations of this command, this version of this command is used to enable you to set up dynamic buffering on serial and parallel ports. The command PRT_USE enables you to connect a buffer to a specified device, altering the description (usage) used to access that buffered device. Initially, the default usage is PRT and the default device is SER which means that any attempt to send output to the PRT device will actually access ser1, using the whole of the available memory as a buffer.

PRT_USE will actually recognise the full device name, allowing it to have a similar effect as the alternative version of this command. For example, the following are both equivalent:

```
PRT_USE prt,ser1c (On the Gold Card)
PRT_BUFF 0: PRT_USE ser1c (Under SMSQ/E)
```

The PRT device will also allow the same options as the device which it is emulating, for example, the following are both the same (except the latter uses buffered output):

```
OPEN #3,ser1c
PRT_USE prt,ser:OPEN #3,prt1c
```

If you wish to buffer output on a given device, then you merely need to specify the usage to be the same as the device. For example:

```
PRT_USE ser,ser
```

will create buffered output to the serial ports whenever ser is used.

PRT_USE will also allow you to specify the device to be buffered at run-time. This is achieved by leaving the device parameter as an empty string. For example:

```
PRT_USE buff_,""
```

allows you to use the device name buff_ser1 to access ser1, buff_par to access the parallel port etc. and all with buffered output.

**NOTE 1**

```
PRT_USE prt,ser
```

will return the QL to the normal state after being switched on (ie. only buffered output will occur if the device PRT is used).

**NOTE 2**

If PRT_USE is used to allow background printing, then some characters may be lost (especially if you are using an old serial to parallel converter), if you use a command which stops the QL multitasking (for example FORMAT, LOAD, LBYTES, SBYTES and SAVE).

You can tell when this happens as the printer will stop while the command is being carried out.

**CROSS-REFERENCE**

See also *PRT_ABT*. See the other version of this command.

## 23.96 PRT_USE$

| | |
|---|---|
| Syntax | PRT_USE$ |
| Location | ST/QL, SMSQ/E |

This function returns a string representing the current port emulated by the PRT device, thus allowing you to check whether or not you need to alter the device set with PRT_USE.

**Example**

```
PRT_USE ser1etf
PRINT PRT_USE$
```

will return 'ser1etf'.

**CROSS-REFERENCE**

See *PRT_USE*.

## 23.97 PTH_ADD

| | |
|---|---|
| Syntax | PTH_ADD [n,] directory |
| Location | Path device |

First we need to explain the PTH device before you can understand what the command PTH_ADD and its related commands/ functions do.

Using sub-directories helps to clean up disk storage - even if you know on which disk a file is kept, if you are using a large storage media like HD/ED disks or even hard disks, you will soon find yourself searching through the whole directory tree with a desktop or WDIR. That's why PTH was created.

This virtual device interfaces with any kind of drive and searches through a list of directories when a file is to be opened. For instance, instead of being forced to type:

```
VIEW win1_games_defender_manual_txt
```

a short:

```
VIEW pth1_manual_txt
```

would be enough to show the manual_txt if the directory win1_games_defender_ is in the path list.

The size of the search list is only limited by memory available; a list of 30000 entries has been tested, 900k was necessary to store it - but this is not a realistic limitation. Who works with several thousand directories?

PTH_ADD modifies that list which can have as many entries as necessary.

PTH_ADD (the name says it already) adds a directory to the path list, it can be inserted (you cannot replace pathnames!) at a certain position by directly specifying the position n (a non-negative integer) in the list.

If n is not specified, the new directory is merely added to the end of the list. The example will clarify this.

**Example**

We assume the path list is empty.

```
PTH_ADD flp1_
```

will add flp1_ to the list which will now look like this (the list can be obtained with:

```
PTH_LIST
```

```
0 flp1_
```

The first column is the number of the entry, programmers tend to start counting with zero, that's why the first entry has the number 0. If you type, for example:

```
SPL pth1_pth_bin,#1
```

the binary file pth_bin in flp1_ will be spooled to channel #1 (usually a window under the interpreter).

Now let's add a few more entries to exploit the power of the path device:

```
PTH_ADD flp1_basic_
PTH_ADD flp2_
PTH_ADD ram1_
```

The list is now:

```
0 flp1_
1 flp1_basic_
2 flp2_
3 ram1_
```

Assume the file myprog_bas is in ram1_:

```
LOAD pth1_myprog_bas
```

tries to load the following files one by one and skips to the next one in case of failure:

- flp1_myprog_bas
- flp1_basic_myprog_bas
- flp2_myprog_bas
- ram1_myprog_bas

If myprog_bas does not appear in any of the directories, the usual 'Not Found' error would appear.

**NOTE 1**

The name of the path device can be freely configured with the Qjump standard configuration program Config (or MenuConfig). We use the default in this manual, which is PTH. The name can be changed temporarily with PTH_USE.

**NOTE 2**

PTH suffers from the same problem as the DEV device, see the note at DEV_USE.

**NOTE 3**

An underscore is added to the directory if it's missing. On this point PTH_ADD behaves differently from DEV_USE.

**WARNING**

Some applications do not co-operate happily with PTH so that a file may get spread over all directories if you save it from some editors etc. There are no crashes, do not worry, but this strange behaviour could lead to a loss of data if you are not aware of the strange phenomenon.

**CROSS-REFERENCE**

The path device is very similar to the dev_ device, please read through *DEV_USE* to understand the idea behind both devices. It's pretty useful to set the Toolkit II *DATAD$* and *PROGD$* to pth1_:

```
DATA_USE pth1_
PROG_USE pth1_
```

Do this preferably in your BOOT program. - Look at the other 'PTH_XXX' keywords starting at *PTH_ADD*!

## 23.98 PTH_LIST

| | |
|---|---|
| Syntax | PTH_LIST [#ch] |
| Location | Path device |

The command PTH_LIST prints a list of the search paths available to the PTH device.

**Examples**

```
PTH_LIST
PTH_LIST#2
```

**CROSS-REFERENCE**

*PTH$* is an alternative way to get the path list.

## 23.99 PTH_RMV

| | |
|---|---|
| Syntax | PTH_RMV n |
| Location | Path device |

This command removes a search path from the search list and all directories below the removed entry are moved up in the list by one position to fill the gap. The number n corresponds to the number in the list produced by PTH_LIST.

**Example**

Assume the following search list:

```
0 flp1_
1 flp1_basic_
2 flp2_
3 ram1_
```

```
PTH_RMV 2
```

will remove entry 2 (flp2_), entry 3 will become entry 2 so that the new list will be:

```
0 flp1_
1 flp1_basic_
2 ram1_
```

The search list can be totally cleaned up with the following little procedure PTH_CLEAR:

```
10 DEFine PROCedure PTH_CLEAR
20   REPeat clean_up
30     IF PTH$(0)="" THEN EXIT clean_up
40     PTH_RMV 0
50   END REPeat clean_up
60 END DEFine PTH_CLEAR
```

**CROSS-REFERENCE**

Other 'PTH_XXX' keywords starting at *PTH_ADD*!

## 23.100 PTH_USE

| | |
|---|---|
| Syntax | PTH_USE [path_name] |
| Location | Path device |

The default name used for the path device is PTH. If you don't like that, you can change it with PTH_USE to any other combination of three letters, including existing drive names. If no parameter is used, the default name is restored.

**Examples**

::

PTH_USE huh PTH_USE flp PTH_USE

**NOTE**

The default name can be permanently configured with Qjump's Config program.

**CROSS-REFERENCE**

*PTH_USE$* returns the current setting.

## 23.101 PTH_USE$

| | |
|---|---|
| Syntax | PTH_USE$ |
| Location | Path device |

As mentioned above, the function PTH_USE$ gives you the name which is used for the path device.

**Example**

```
PRINT PTH_USE$
```

## 23.102 PTH$

| | |
|---|---|
| Syntax | PTH$ (n) |
| Location | Path device |

The function PTH$ returns the nth directory in the search list of the path device.

**Examples**

The procedure PTH_INFO prints all of the current settings concerning the the pth device to #1.

The function PTH_ENTRIES% returns the number of directories in the path list.

```
100 DEFine PROCedure PTH_INFO
110   LOCal n: n=0
120   PRINT "Path device:"!PTH_USE$
130   PRINT\"Search paths";
140   REPeat list_them
150     IF PTH$(n)="" THEN EXIT list_them
160     IF NOT n THEN PRINT
170     PRINT FILL$(" ",3-LEN(n));n;TO 5,PTH$(n)
180     n=n+1
190   END REPeat list_them
200   IF NOT n THEN PRINT " no entries"
210 END DEFine PTH_INFO
220 :
240 DEFine FuNction PTH_ENTRIES%
250   LOCal n
260   FOR n=0 TO 32767: IF PTH$(n)="" THEN EXIT n
270   RETurn n
280 END DEFine PTH_ENTRIES%
```

**CROSS-REFERENCE**

*PTH_ADD*, *PTH_LIST*

## 23.103 PTR_FN%

| | |
|---|---|
| Syntax | PTR_FN% (offset) |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I - v2.8+) |

The DIY Toolkit includes code which allows you to link a serial mouse to the QL, similar to the commercial SERMouse package which is packaged with SMSQ/E for the Gold Card. Refer to the Appendix on Mice for more details.

The mouse is enabled with PTR_ON. This function can be used to read various values used by the DIY mouse driver and which can be altered using other commands from this toolkit.

The value of offset should be in the range 0…13 and returns the following:

| Offset | Meaning |
|---|---|
| 0 | Latest X position (Read with X_PTR%) |
| 1 | Latest Y position (Read with Y_PTR%) |
| 2 | Maximum X co-ordinate (Set with PTR_MAX) |
| 3 | Maximum Y co-ordinate (Set with PTR_MAX) |
| 4 | Step X (Set with PTR_INC) |
| 5 | Step Y (Set with PTR_INC) |
| 6 | Details of buttons pressed (Read with BUTTON%) |
| 7 | Synchronisation counter (Read with SYNCH%) |
| 8 or 9 | Zero, or serial channel ID |
| 10 | Accumulated X drift |
| 11 | Accumulated Y drift |
| 12 | Set = cursor key emulation (Set with PTR_KEY) |
| 13 | Set = Pointer Wrap (Set with PTR_KEY) |

The Accumulated X and Y drift are counters, used by the serial mouse driver to judge how far off the horizontal / vertical the mouse has moved and whether to continue moving the pointer in a straight line or to take this into account.

**CROSS-REFERENCE**

For more details, refer to the individual commands/ functions.

## 23.104 PTR_INC

| | |
|---|---|
| Syntax | PTR_INC x_step, y_step |
| Location | KMOUSE (DIY Toolkit - Vol I) |

This command is only really of any use when the Cursor Key emulation is enabled (see PTR_KEY). It allows you to set the number of mouse pulses which are taken to correspond to moving the cursor 1 character either in an x direction or a y direction.

The two values given are normally set to 12 and 24 respectively for MODE 4 operation, although if this proves too quick (especially in MODE 8), you could try PTR_INC 24,24. The higher the values, the slower the cursor will move as you push the mouse about the table.

**CROSS-REFERENCE**

See *PTR_KEY* and also compare *SERMSPEED*.

## 23.105 PTR_KEY

| | |
|---|---|
| Syntax | PTR_KEY cursor, edge |
| Location | KMOUSE (DIY Toolkit - Vol I) |

Normally DIY Toolkit's mouse driver will enable you to control the mouse pointer on screen. This mouse pointer is however, not the one used by the Pointer Environment (therefore the mouse cannot be used to control programs written specifically for the Pointer Environment except in cursor emulation mode) and

you need a separate program to run in the background which will display a symbol to show the position of the mouse on screen.

Th PTR_KEY command allows you to specify whether the serial mouse driver should emulate the cursor keys (instead of the pointer), which allows it to operate software such as word processors.

To emulate the cursor keys, cursor should be 1 - to emulate the pointer again, set cursor to 0.

The DIY Toolkit mouse driver is actually better than the SERMouse driver in this respect in that the mouse does not automatically switch back into Pointer Mode when you leave the program (see SERMCUR). Then again, you cannot switch between the two modes using the mouse buttons, or control Pointer Environment programs…

The second parameter expected by this command is used to specify what should happen to the cursor (or pointer) at the edge of the screen - if edge=1, moving the cursor or pointer over the edge of the screen will make it re-emerge on the opposite edge (a wrapping effect). edge=0 disables this.

**CROSS-REFERENCE**

See *PTR_INC* also. Also see *PTR_ON* and *SERMPTR*

## 23.106 PTR_LIMITS

| | |
|---|---|
| Syntax | PTR_LIMITS minx, miny, maxx, maxy |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I), Amiga QDOS (v3.20+) |

This command is used to set the limits of the screen over which the mouse pointer can be moved using the mouse. The command expects four parameters, the minimum x and y co-ordinates and the maximum x and y co-ordinates. For a standard QL, you would normally set these values with:

```
PTR_LIMITS 0,0,511,255
```

However, on larger resolution displays, larger limits will be needed.

On the DIY Toolkit variant, the first two limits are ignored (they are always taken to be zero). The maximum co-ordinates should be in pixel sizes and can be any positive number up to 32767.

On Amiga QDOS, negative numbers can be used, but to retain compatibility, the first two parameters should be zero.

Having set these parameters, once the mouse pointer has reached this position on screen then what happens depends on whether the wrap-around display mode has been enabled with PTR_KEY 0,1 or PTR_KEY 1,1 (or not). If it has been disabled, then the mouse pointer will move no further. If it has been enabled, then the mouse pointer will appear at the other extreme limit.

**CROSS-REFERENCE**

*PTR_POS* can be used to dictate where the mouse pointer should appear on screen. This command only calls *PTR_MAX* on the DIY implementation.

## 23.107 PTR_MAX

| | |
|---|---|
| Syntax | PTR_MAX maxx, maxy |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I), Amiga QDOS v3.20+ |

This command is the same as: PTR_LIMITS 0,0,maxx,maxy

**CROSS-REFERENCE**

See *PTR_LIMITS*!

## 23.108 PTR_OFF

| | |
|---|---|
| Syntax | PTR_OFF |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I), Amiga QDOS v3.20+ |

This command switches off the mouse driver, releasing memory which is used by it for temporary shortage. PTR_ON switches the driver back on.

**CROSS-REFERENCE**

This is similar to *SERMOFF*.

## 23.109 PTR_ON

| | |
|---|---|
| Syntax | PTR_ON |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I), Amiga QDOS v3.20+ |

This command enables the mouse driver after it has been loaded or after it has been disabled with PTR_OFF. All of the mouse settings are reset to the defaults (set when the files were originally assembled) and the pointer is positioned in the top left corner of the screen (position 0,0).

**CROSS-REFERENCE**

You can re-position the mouse with *PTR_POS*. See *SERMON* and *PTR_OFF*. *PTR_INC*, *PTR_KEY* and *PTR_MAX* are also needed to set various parameters on start-up.

## 23.110 PTR_POS

| | |
|---|---|
| Syntax | PTR_POS x,y |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I), Amiga QDOS v3.20+ |

This command can be used to set the initial position of the mouse pointer on screen - it is normally located at 0,0 (the top left hand corner of the screen).

You can however use this command to set it to the specified absolute pixel co- ordinates, which must be within the area defined with the PTR_LIMITS command.

**CROSS-REFERENCE**

See *PTR_LIMITS* and *PTR_ON*. The pointer position can be read with *X_PTR%*, *Y_PTR%* and *PTR_FN%*.

## 23.111 PTR_X

| | |
|---|---|
| Syntax | PTR_X (argument, module) |
| Location | PTRRTP |

The function PTR_X transforms a point (described in polar co-ordinates) into the rectangular co-ordinates and returns the real part of the latter. argument is an angle in radians, module a radius.

**Example**

A line in rectangular co-ordinates transformed to polar co-ordinates looks like a circle when plotted on screen. However, if you were to look at this line using polar co-ordinates, it would appear as straight line again. Confused?

```
100 SCALE 10,-5,-5: PAPER 0: CLS
110 FOR a = 0 TO 2*PI STEP PI/128
120   POINT PTR_X(a,2), PTR_Y(a,2)
130 END FOR a
```

**CROSS-REFERENCE**

The other rectangular co-ordinate is found with *PTR_Y* below. Also see *RTP_R* and *RTP_T*

## 23.112 PTR_Y

| | |
|---|---|
| Syntax | PTR_Y (argument, module) |
| Location | PTRRTP |

This function is very similar to PTR_X but this time the imaginary part is returned.

## 23.113 PURGE

| | |
|---|---|
| Syntax | PURGE |
| Location | CONTROL (DIY Toolkit Vol E) |

The command PURGE has the same (fatal) effect as KILL or KJOBS.

## 23.114 PUT

| | |
|---|---|
| Syntax | PUT [#channel\file_position,] [item$^1$ $^*$[,item$^i$]$^*$ ...] or<br>PUT [#channel,] [item$^1$ $^*$[,item$^i$]$^*$ ...] |
| Location | Toolkit II, THOR XVI |

This command forms the complement to GET and allows you to store variables in the specified channel (default #3) in the QL's internal format.

The variables are stored at the current position in the file (or the file_position given with the command, if the first variant is used).

If you provide more than one variable name as the second, third parameter etc, then several variables will be stored in the file in one go.

If no variable is specified, the file pointer will be set to the specified position if the first variant is used.

If the second variant is used, this will have no effect.

If a variable is given as the file pointer, then this variable will be updated with the current file position once PUT has finished its work.

PUT can actually be used to store variables in a different type to their current use (this might, for example, be useful if storing part of a string), by adding the following suffixes to each item:

| | |
|---|---|
| +0 | Force floating point type (see note 2 below!) |
| &” | Force string type |
| \|\|0 | Force integer type |

**Example**

```
a$='Entry 123':PUT #3,a$,a$(7 TO) || 0
```

will store in channel #3 two bytes giving the length of the string a$, followed by the characters of the string itself, followed by two bytes representing the integer value 123 (ie. 0*256+123).

**NOTE 1**

On version 2.09 (or earlier) of Toolkit II, PUT could cause problems when used on a channel opened over the network.

**NOTE 2**

Although PUT can convert variable types as above, if integer tokenisation is enabled on Minerva, then PUT x%+0 will not work. You need to use something such as PUT x%+1e-555 instead.

**NOTE 3**

Whenever storing data on disk, it is always preferable to store it in its internal format (unless it is to be read on other systems as well). Internal storage is faster because conversion between internal and readable format is no longer necessary. It also produces shorter files since the internal format needs less space and for floating point numbers, the internal format gives the greatest possible accuracy.

**CROSS-REFERENCE**

See *PUT*, *BPUT*, *WPUT*, *LPUT*, *UPUT* and *GET*.

## 23.115 PUT_BYTE

| | |
|---|---|
| Syntax | PUT_BYTE #channel, byte |
| Location | DJToolkit 1.16 |

The given byte is sent to the channel. If a byte value larger than 255 is given, only the lowest 8 bits of the value are sent. The byte value written to the channel will always be between 0 and 255 even if a negative value is supplied. *GET_BYTE* returns all values as positive.

**EXAMPLE**

```
PUT_BYTE #3, 10
```

**CROSS-REFERENCE**

*PUT_FLOAT*, *PUT_LONG*, *PUT_STRING*, *PUT_WORD*.

## 23.116 PUT_FLOAT

| | |
|---|---|
| Syntax | PUT_FLOAT #channel, byte |
| Location | DJToolkit 1.16 |

The given float value is converted to the internal QDOS format for floating point numbers and those 6 bytes are sent to the given channel number. The full range of QL numbers can be sent including all the negative values. *GET_FLOAT* will return negative values correctly (unless an error occurs).

**EXAMPLE**

```
PUT_FLOAT #3, PI
```

**CROSS-REFERENCE**

*PUT_BYTE*, *PUT_LONG*, *PUT_STRING*, *PUT_WORD*.

## 23.117 PUT_LONG

| | |
|---|---|
| Syntax | PUT_LONG #channel, byte |
| Location | DJToolkit 1.16 |

The long value given is sent as a sequence of four bytes to the channel. Negative values can be put and these will be returned correctly by *GET_LONG* unless any errors occur.

**EXAMPLE**

```
PUT_LONG #3, 1234567890
```

**CROSS-REFERENCE**

*PUT_BYTE*, *PUT_FLOAT*, *PUT_STRING*, *PUT_WORD*.

## 23.118 PUT_STRING

| | |
|---|---|
| Syntax | PUT_STRING #channel, string |
| Location | DJToolkit 1.16 |

The string parameter is sent to the appropriate channel as a two byte word giving the length of the data then the characters of the data. If you send a string of zero length, LET A$ = “” for example, then only two bytes will be written to the file. See *POKE_STRING* for a description of what will happen if you

supply a number or a numeric variable as the second parameter. As with all QL strings, the maximum length of a string is 32kbytes.

**EXAMPLE**

```
PUT_STRING #3, "This is a string of data"
```

**CROSS-REFERENCE**

*PUT_BYTE*, *PUT_FLOAT*, *PUT_LONG*, *PUT_WORD*.

## 23.119 PUT_WORD

| | |
|---|---|
| Syntax | PUT_WORD #channel, word |
| Location | DJToolkit 1.16 |

The supplied word is written to the appropriate channel as a sequence of two bytes. If the word value supplied is bigger than 65,535 then only the lower 16 bits of the value will be used. Negative values will be returned by *GET_WORD* as positive.

**EXAMPLE**

```
PUT_WORD #3, 65535
```

**CROSS-REFERENCE**

*PUT_BYTE*, *PUT_FLOAT*, *PUT_LONG*, *PUT_STRING*.

## 23.120 PXOFF

| | |
|---|---|
| Syntax | PXOFF |
| Location | PEX |

This command is the same as PIE_EX_OFF except for the PEX system extension - it disables PEX for SD.EXTOP system calls, so that they are trapped by the Pointer Environment.

**CROSS-REFERENCE**

See *PXON* and*PEON*.

## 23.121 PXON

| | |
|---|---|
| Syntax | PXON |
| Location | PEX |

This command is the same as PIE_EX_ON except for the PEX system extension - it enables PEX for SD.EXTOP system calls, so that they can work in the background.

**CROSS-REFERENCE**

See *PXOFF* and*PEON*.

## 23.122 PX1ST

| | |
|---|---|
| Syntax | PX1ST [{ flag }] |
| Location | PEX |

This function can be used to determine whether IS_PTRAP has been used to dictate that any screen operations should just be ignored (this is the default under PEX on JS and MG ROMs). The function name has a digit one in it, not a letter 'eye'.

If the value returned is 0, then (providing that you do not have a JS or MG ROM), PEX may be allowing background screen access.

If the value is not 0, then screen operations are merely ignored by the operating system (and therefore the display is not affected).

If you use this function to pass a parameter, then if the parameter is 0, then screen operations will not be ignored and whether they cause a Buried program to halt will depend on whether PEX is active. If you pass a non-zero parameter, then any screen operations will be ignored.

**CROSS-REFERENCE**

See *PEON* and in particular, refer to *IS_PTRAP* for more details. See also *IS_PEON*.

## 23.123 P_ENV

| | |
|---|---|
| Syntax | P_ENV (#ch) |
| Location | MULTI |

This function detects whether the given channel is running under the Pointer Environment and returns:

- 0 if the Pointer Environment is not connected to that channel, or no parameter was used, or #ch is not a screen channel;
- 1 if the Pointer Interface (ptr_gen) is active in that channel;
- 2 if the Pointer Interface and the Window Manager (wman) are present for that channel.

**Example**

All programs which need the Pointer Environment to work, should check to see if it is present. This short program does so:

```
100 ch=FOPEN(con_2x2a0x0): p=P_ENV(#ch): CLOSE#ch
110 IF p<2 THEN
120   PRINT "This program does not run without the P.E.,"
130   PRINT "so program execution has to stop here."
140   PRINT " Press any key...": PAUSE 400
150   STOP
160 END IF
```

**CROSS-REFERENCE**

*WMAN$* finds the version of the Window Manager and *QRAM$* the version of the Pointer Interface. These functions can also be used to detect the presence of the Pointer Environment (=Window Manager & Pointer Interface).

CHAPTER

TWENTYFOUR

# KEYWORDS Q

## 24.1 QACONVERT

| | |
|---|---|
| Syntax | QACONVERT filename |
| Location | ATARIDOS |

This command takes a file which is stored on a QL Format disk and converts it into Atari Format. It will then convert special characters in that file to Atari compatible characters as well as converting any occurrence of a Line Feed character CHR$(10) to a Carriage Return character CHR$(13) followed by a Line Feed character CHR$(10).

**CROSS-REFERENCE**

Compare *AQCONVERT* and *QICONVERT*. See also *ACOPY* and *QCOPY*.

## 24.2 QCOPY

| | |
|---|---|
| Syntax | QCOPY filename1,filename2 |
| Location | ATARIDOS |

This command is similar to COPY except that it copies a file from an Atari Format disk to a QL Format disk. No conversion takes place.

**NOTE**

You will need to pass the Atari filename in quote marks if it includes a three letter extension preceded by a dot eg:

```
QCOPY "flp1_PROGRAM.BAS", flp2_PROGRAM.BAS
```

**CROSS-REFERENCE**

*ACOPY* copies a file from a QL disk to an Atari disk. Level-3 device drivers allow you to read and write to Atari and IBM format disks anyway. See *AFORMAT* and *AQCONVERT*.

## 24.3 QCOUNT%

| | |
|---|---|
| Syntax | QCOUNT% (#pipe_ch) |
| Location | PIPES (DIY Toolkit - Vol P) |

This function is designed to make the use of pipes easier. Provided that the specified channel is linked to a pipe (normally the input end of the pipe), this function will return the number of characters which are waiting to be read from the pipe.

**NOTE**

The value returned will be the exact number of characters placed into the pipe, which will therefore include the line feed characters and other control characters. Take the example of:

```
10 OPEN #3,pipe_1048
20 PRINT #3,'HELLO'
30 PRINT QCOUNT%(#3)
```

The value of 6 is returned, this is because the PRINT statement has added a line feed to the end of the word 'HELLO', so that this can be read by an INPUT command.

**CROSS-REFERENCE**

Refer to *QLINK* and also the Device Drivers Appendix. See also *QSIZE%* and *QSPACE%*.

## 24.4 QDOS$

| | |
|---|---|
| Syntax | QDOS$ |
| Location | Fn, TinyToolkit, BTool |

This function returns a string containing the version code of the operating system on which SuperBASIC is running. For example

```
PRINT QDOS$
```

- Version 1.03 was the first main version of QDOS (an upgrade is essential if you have an earlier version!)
- Version 1x13 is for all MGx ROMs, eg. 1G13 for the MGG.
- 1.63 was the first version of Minerva.
- 1.76 was the first version of Minerva with reliable MultiBASICs.
- 1.98 was the latest and best version of Minerva
- 2.xx SMS version.
- 3.xx is the version number of the Amiga QL Emulator.
- 4.xx is the first version of ARGOS on a THOR 1 computer. #
- 5.xx is the version of ARGOS on a THOR 20 computer.
- 6.xx is the version of ARGOS on a THOR XVI computer.

- 6.41 is the final version of the THOR ARGOS ROM.

**NOTE 1**

VER$ is normally used to identify the version of the SuperBASIC interpreter and QDOS$ to identify the version of QDOS (the operating system). However, as SuperBASIC is an integral part of the operating system on most QDOS computers, there should really only be a need to use one of these functions and not both.

**NOTE 2**

The '.' in QDOS$ is changed on MG and SMS v2.50+ (on Gold Cards and Super Gold Cards only) to reflect the country code of the language version currently loaded.

**CROSS-REFERENCE**

*VER$* contains another code identifying the operating system. *VER$*(1) is identical to *QDOS$* for Minerva ROMs and SMS. You should also look at *MACHINE* and *PROCESSOR*. *LANG_USE* allows you to change the operating system language.

## 24.5 QFLIM

| | |
|---|---|
| Syntax | QFLIM ([#channel,] n) n=0..3 |
| Location | Fn |

With the Pointer Interface present, each job has a maximum outline window size in which it can open its windows to avoid storing more information than necessary when switching between jobs and saving the window contents. The function QFLIM returns the following information about this maximum outline size, in the (window independent) absolute co-ordinate system for the different n:

| N | Information Returned |
|---|---|
| 0 | Width in pixels (eg. 512 on a standard QL display). |
| 1 | Height in pixels (eg. 256 on a standard QL display). |
| 2 | Leftmost horizontal position. |
| 3 | Uppermost vertical position. |

QFLIM needs an open window to get the information from (default #1). The return values refer to the current job. If any other values of n are used, a bad parameter error will be generated.

**Example**

If there are no windows other than #0, #1 and #2, and their positions are set up as follows:

```
100 WINDOW #0,100,100,50,50
110 WINDOW #1,20,20,0,0
120 WINDOW #2,200,50,40,40
130 FOR c=0 TO 2: PAPER #c,3: BORDER #c,1,4: CLS #c
```

The Pointer Interface will reduce the outline size of the screen available to SuperBASIC, which can be checked with the next program or by swapping to other jobs which fill the whole screen.

```
100 ch=2
110 xmin = QFLIM(#ch, 2): xmax = xmin + QFLIM(#ch,0)
120 ymin = QFLIM(#ch, 3): ymax = ymin + QFLIM(#ch,1)
130 PRINT "x = "; xmin; ".."; xmax
140 PRINT "y = "; ymin; ".."; ymax
150 percent% = 100 * QFLIM(#ch, 0) * QFLIM(#ch, 1) / (512 * 256)
160 PRINT "fills"! percent%; "% of the screen"
```

Type WTV or WMON to restore standard window sizes. Note that this example expects the display to be 512x256 pixels.

**NOTE 1**

The Pointer Interface makes a distinction between the primary window (generally the first window to be used for input/output operations) and secondary windows. Although using QFLIM on a secondary window will return the maximum outlines for the current job's windows, using QFLIM on the primary window (eg. #0 in SuperBASIC) will return the physical screen size, ie. the parameters of the largest possible window:

```
WINDOW QFLIM(#0, 0), QFLIM(#0, 1), QFLIM(#0, 2), QFLIM(#0, 3)
```

This can therefore be used to check whether or not the extra high resolution modes provided by some Emulators and the AURORA is available:

```
exten4 = 0
IF QFLIM(#0,1) > 256: exten4 = 1
```

**NOTE 2**

QFLIM returns useless numbers greater than 10000 if the Pointer Interface is not present.

**CROSS-REFERENCE**

*WMAN$*, *WINF$* allow you to find out various information about the Pointer Environment. *XLIM*, *SCR_XLIM*, *YLIM* and *SCR_YLIM* are similar to *QFLIM*. See also *OUTLN*.

## 24.6 QICONVERT

| | |
|---|---|
| Syntax | QICONVERT filename |
| Location | ATARIDOS |

This command takes a file which is stored on a QL Format disk and converts it into IBM Format. It will then convert special characters in that file to IBM compatible characters as well as converting any occurrence of a Line Feed character CHR$(10) to a Carriage Return character CHR$(13) followed by a Line Feed character CHR$(10).

**CROSS-REFERENCE**

Compare *IQCONVERT* and *AQCONVERT*. See also *ACOPY* and *QCOPY*. See *IFORMAT*.

## 24.7 QLINK

| | |
|---|---|
| Syntax | QLINK #output TO #input |
| Location | PIPES (DIY Toolkit - Vol P) |

This command is the same as TCONNECT.

**CROSS-REFERENCE**

The following functions are also useful when accessing pipes: *EOFW*, *PEND*, *QSIZE%*, *QCOUNT%* and *QSPACE%*.

## 24.8 QLOAD

| | |
|---|---|
| Syntax | QLOAD [device_]filename |
| Location | SMS |

This command is very similar to LOAD as implemented on the SMS. The only difference are that it insists that the program must have been saved with the _sav suffix (eg. flp1_TEST_sav). QLOAD will then proceed to load the BASIC program whether it was saved with the normal SAVE or SAVE_O commands, or with the QSAVE or QSAVE_O commands.

**NOTE 1**

If a program has been saved using QSAVE on a Minerva machine with Integer Tokenisation enabled, then QLOAD will not be able to understand it properly and you will notice that numbers and keywords have been replaced by various symbols.

**NOTE 2**

Any commands which appear after a QLOAD command will be ignored.

**CROSS-REFERENCE**

Also see *LOAD*, *QLRUN*, *QMERGE* and *QSAVE* Compare *UNLOAD*.

## 24.9 QLRUN

| | |
|---|---|
| Syntax | QLRUN [device_]filename |
| Location | SMS |

This command is exactly the same as QLOAD except that the program is automatically RUN as soon as it has been loaded into memory.

**CROSS-REFERENCE**

See *QLOAD* and *QMRUN*.

## 24.10 QL_PEX

| | |
|---|---|
| Syntax | QL_PEX |
| Location | PEX |

This function returns the offset of the keyword linkage block of the keywords added by the PEX toolkit. This offset is needed for Qliberator's $$asmb directorive.

**CROSS-REFERENCE**

See *PEX_SAVE*.

## 24.11 QMERGE

| | |
|---|---|
| Syntax | QMERGE [device_]filename |
| Location | SMS |

This command bears the same relationship to MERGE as QLOAD does to LOAD.

**CROSS-REFERENCE**

Refer to *QLOAD* and *MERGE*. See also *QMRUN*

## 24.12 QMRUN

| | |
|---|---|
| Syntax | QMRUN [device_]filename |
| Location | SMS |

This command is exactly the same as QMERGE except that it ensures that the program is RUN as soon as it has been merged into memory. If the command is issued from the command line as a direct command, then the merged program is RUN from line 1. If, however, QMRUN appears in the program itself, the program continues from the statement following QMRUN (making it the same as QMERGE).

**CROSS-REFERENCE**

See *QMERGE* and *MRUN*.

## 24.13 QPC_CMDLINE$

| | |
|---|---|
| Syntax | cmd$ = QPC_CMDLINE$ |
| Location | SMSQ/E for QPC |

This returns the argument that was supplied to QPC after the "-cmdline" command line argument. This can be used to do different actions depending on the way QPC was started.

## 24.14 QPC_EXEC

| | |
|---|---|
| Syntax | QPC_EXEC command$[, parameter$] |
| Location | SMSQ/E for QPC |

This command can be used to call an external DOS or Windows program. The name of the executable file is given in the first parameter. Optionally, you can also supply a second parameter, which is then passed to the executed program as its command line arguments.

Furthermore, you can supply a data file as the first parameter. In this case, the associated application for this file type is executed.

**Example**

```
QPC_EXEC 'notepad','c:\text.txt'
```

Starts notepad and loads the c:\text file.

```
QPC_EXEC 'c:\text.txt'
```

Starts the default viewer for .txt files.

## 24.15 QPC_EXIT

| | |
|---|---|
| Syntax | QPC_EXIT |
| Location | SMSQ/E for QPC |

This simply quits QPC.

## 24.16 QPC_HOSTOS

| | |
|---|---|
| Syntax | os% = QPC_HOSTOS |
| Location | SMSQ/E for QPC |

This function returns the host operating system under which QPC was started.

Possible return codes are:

- 0 = DOS (QPC1)
- 1 = Win9x/ME (QPC2)
- 2 = WinNT/2000/XP (QPC2)

## 24.17 QPC_MAXIMIZE

| | |
|---|---|
| Syntax | QPC_MAXIMIZE |
| Location | SMSQ/E for QPC |

Maximises the QPC window. (Yes, the spelling of the command name is American!)

## 24.18 QPC_MINIMIZE

| | |
|---|---|
| Syntax | QPC_MINIMIZE |
| Location | SMSQ/E for QPC |

Minimizes the QPC window. (Yes, the spelling of the command name is American!)

## 24.19 QPC_MSPEED

| | |
|---|---|
| Syntax | QPC_MSPEED x_accel, y_accel |
| Location | SMSQ/E for QPC |

This command has no effect on QPC2.

## 24.20 QPC_NETNAME$

| | |
|---|---|
| Syntax | name$ = QPC_NETNAME$ |
| Location | SMSQ/E for QPC |

This function returns the current network name of your PC (the one you supplied upon installation of Windows). The result can be used to distinguish between different PCs (For example, in a BOOT program).

## 24.21 QPC_QLSCREMU

| | |
|---|---|
| Syntax | QPC_QLSCREMU value |
| Location | SMSQ/E for QPC |

Enables or disables the original QL screen emulation. When emulating the original screen, all memory write accesses to the area $20000-$207FFF are intercepted and translated into writes to the first 512x256 pixels of the big screen area. If the screen is in high colour mode, additional colour conversion is done.

Possible values are:

- -1: automatic mode
- 0: disabled (default)
- 4: force to 4-colour mode
- 8: force to 8-colour mode

When in QL colour mode, the emulation just transfers the written bytes to the larger screen memory, i.e. when the big mode is in 4-colour mode, the original screen area is also treated as 4-colour mode. In high colour mode however, the colour conversion can do both modes. In this case, you can pre-select the

emulated mode (parameter = 4 or 8) or let the last issued *MODE* call decide (automatic mode). Please note that that automatic mode does not work on a per-job basis, so any job that issues a *MODE* command changes the behaviour globally.

Please also note that this transition is one-way only, i.e. bytes written legally to the first 512x256 pixels are not transferred back to the original QL screen (in the case of a high colour screens this would hardly be possible anyway). Unfortunately, this also means that not all old programs will run perfectly with this type of emulation. If you experience problems, start the misbehaving application in 512x256 mode.

## 24.22 QPC_RESTORE

| | |
|---|---|
| Syntax | QPC_RESTORE |
| Location | SMSQ/E for QPC |

Restores the QPC window. This will return the window size from minimised or maximised to what it was before.

## 24.23 QPC_SYNCSCRAP

| | |
|---|---|
| Syntax | QPC_SYNCSCRAP |
| Location | SMSQ/E for QPC |

In order to rapidly exchange text passages between Windows and SMSQ/E the Syncscrap functionality has been introduced. The equivalent of the Windows clipboard is the scrap extension of the menu extensions.

After loading the menu extensions you can call this command, which creates a job that periodically checks for changes in either the scrap or the Windows clipboard, and synchronizes their contents if necessary. Please note that only text data is supported. The character conversion between the QL character set and the Windows ANSI set is done automatically. The line terminators (LF or LF+CR) are converted too.

## 24.24 QPC_VER$

| | |
|---|---|
| Syntax | v$ = QPC_VER$ |
| Location | SMSQ/E for QPC |

This returns the current QPC version.

**Example**

```
PRINT QPC_VER$
```

Will print 4.00 or higher.

## 24.25 QPC_WINDOWSIZE

| | |
|---|---|
| Syntax | QPC_WINDOWSIZE x, y |
| Location | SMSQ/E for QPC |

This sets the size of the client area (the part that displays SMSQ/E) of the QPC window. It does NOT alter the resolution SMSQ/E runs with, so the pixels are effectively zoomed. It is equivalent to the "window size" option in the main configuration window. If QPC is currently in full screen mode it will switch to windowed mode. Window size cannot be set smaller than the SMSQ/E resolution or bigger than the desktop resolution.

**Example**

```
DISP_SIZE 512,256
QPC_WINDOWSIZE 1024,512
```

Does a 200% zoom of the QPC window.

## 24.26 QPC_WINDOWTITLE

| | |
|---|---|
| Syntax | QPC_WINDOWTITLE title$ |
| Location | SMSQ/E for QPC |

Sets the string that can be seen when QPC runs in windowed mode. This can be used to easily distinguish between several QPC instances.

**Example**

```
QPC_WINDOWTITLE "Accounting"
```

Sets the title to "Accounting", without the quotes though!

## 24.27 QPTR

| | |
|---|---|
| Syntax | PE_Found = QPTR(#channel) |
| Location | DJToolkit 1.16 |

This function returns 1 if the Pointer Environment is loaded or 0 if not. The channel must be a SCR_ or CON_ channel, if not, the result will be 0. If a silly value is given then a QDOS error code will be returned instead.

**EXAMPLE**

```
PRINT QPTR(#0)
```

will print 1 of the PE is loaded or zero otherwise.

## 24.28 QRAM$

| | |
|---|---|
| Syntax | QRAM$ |
| Location | TinyToolkit, BTool |

This function returns a string containing the version number of the Pointer Environment, or an empty string if this is not present.

**CROSS-REFERENCE**

*PINF$* is exactly the same as *QRAM$*. *WMAN$* and *WINF$* contain the version number of the Window Manager.

## 24.29 QSAVE

| | |
|---|---|
| Syntax | QSAVE [device_]filename or<br>QSAVE |
| Location | SMS |

For several years now, the best utility for saving SuperBASIC programs in a form which can be loaded very quickly into memory has been QLOAD from Liberation Software.

This utility stores SuperBASIC programs on disk in a special format which although seems meaningless if you VIEW the file, allows the program to be loaded at around 3x the speed of the normal LOAD command, which can be very useful for large programs.

Unlike other similar utilities, programs which have been saved using this utility can be loaded into any other ROM version without any trouble, using the QLOAD command. It is nice to see that this utility has been implemented as part of SMS.

The QSAVE command allows you to save the whole of SuperBASIC program currently in memory under the specified filename to the specified device. If the filename does not end in the suffix _SAV, then this will be added automatically.

If no device is specified (or it does not exist), then Toolkit II's default data device will be used. You will also be prompted to confirm whether an existing file should be overwritten if necessary.

The second variant of the command will allow you to QSAVE the program in memory under the same filename as when LOAD or QLOAD was last used (with the _SAV suffix appended if necessary).

If the original filename used when the program was LOADed ended in _BAS, then QSAVE will alter this to be the _SAV suffix.

This variant will also take the version number of the file when it was LOADed (or QLOADed) and then increase this by one.

**NOTE 1**

To ensure that QSAVEd programs can be used on all implementations of the QL, ensure that if used from Minerva, Integer Tokenisation is switched off - you will need to follow the following procedure:

1. POKE \\212,128
2. LOAD the ASCII version of the program (or type NEW)
3. Alter the program as necessary …
4. QSAVE the fast loading version of the program.

**NOTE 2**

QSAVE without a filename suffers the same problems as SAVE.

**CROSS-REFERENCE**

See *SAVE*, *QLRUN* and *QMERGE*. *DATAD$* allows you to read the current default data device. See also *QSAVE_O*. *FVERS* allows you to read the current version number of a file.

## 24.30 QSAVE_O

| | |
|---|---|
| Syntax | QSAVE_O [device_]filename or QSAVE_O |
| Location | SMS |

This command is the same as QSAVE except that it will automatically overwrite an existing file with the same filename.

**NOTE**

On Minerva machines you need to be careful about Integer Tokenisation - see QSAVE.

**CROSS-REFERENCE**

See *QSAVE*.

## 24.31 QSIZE%

| | |
|---|---|
| Syntax | QSIZE% (#pipe_ch) |
| Location | PIPES (DIY Toolkit - Vol P) |

This function is designed to read the amount of characters which a pipe linked to the specified channel can hold at any one time.

**Example**

```
10 OPEN #4,pipe_200
20 QLINK #4 TO #3
25 PRINT #4,'QL DATA'
30 PRINT QSIZE% (#3), QCOUNT%(#3)
40 CLOSE #3: CLOSE #4
```

This short program will print 203 and 8 on screen.

This will also work with named pipes on SMS:

```
10 OPEN_NEW #4,pipe_test_200
20 OPEN_IN #3,pipe_test
25 PRINT #4,'QL DATA'
30 PRINT QSIZE% (#3), QCOUNT%(#3)
40 CLOSE #3:CLOSE #4
```

Note however, that if you re-run the program the figure returned by QCOUNT% continues increasing - this is because a named pipe does not disappear just because both ends of the pipe have been closed. You would need to add the line:

```
50 DELETE pipe_test
```

to overcome this. Alternatively, try:

```
DIR pipe: WDEL pipe
```

**NOTE**

A pipe can normally hold a few extra characters that the size originally given to the pipe (in the example 203 is returned on most implementations rather than 200 as might be expected). This does not cause a problem.

**CROSS-REFERENCE**

Refer to *QLINK* and also the Device Drivers Appendix. See also *QCOUNT%* and *QSPACE%*.

## 24.32 QSPACE%

| | |
|---|---|
| Syntax | QSPACE% (#pipe_ch) |
| Location | PIPES (DIY Toolkit - Vol P) |

This function returns the amount of empty space in a pipe connected to the specified channel.

```
PRINT QSPACE(#3)
```

is therefore the same as:

```
PRINT QSIZE%(#3) - QCOUNT%(#3)
```

**CROSS-REFERENCE**

Refer to *QCOUNT%* and *QSIZE%*.

## 24.33 QTRAP

| | |
|---|---|
| Syntax | QTRAP #ch,key [,d1 [,d2 [,d3 [,a1 [,a2 ]]]]] |
| Location | TRAPS (DIY Toolkit Vol T) |

This command is similar to IO_TRAP in that it allows you to access the machine code TRAP #3 system calls directly.

You will need to pass at least two parameters, the number of the channel to be affected and the operation key to be carried out (this is equivalent to the value in D0 when TRAP #3 is performed).

The other parameters allow you to pass the various register values which may be required by the system calls. The timeout parameter (D3) defaults to -1 (infinite timeout).

This can be used effectively to set the INK and PAPER colours for THOR XVI's MODE 12 and still allow the program to be compiled. For example:

```
QTRAP #2,HEX('27'),4
```

will set the PAPER colour in the window #2 to Green (although the STRIP colour will remain unaffected).

**WARNING**

Several TRAP #3 calls can crash the computer - make certain that you know what you are doing!

**CROSS-REFERENCE**

See *IO_TRAP*, *TTET3*, *MTRAP* and *BTRAP*. Any return parameters can be read with *DATAREG* and *ADDREG*. *CLS*, *PAN* and *SCROLL* can also be used to call TRAP #3. Refer to the QDOS/SMS Reference Manual (Section 15) for details of the various system TRAP #3 calls.

## 24.34 QuATARI

| | |
|---|---|
| Syntax | QuATARI |
| Location | Beuletools, FN |

This is a logical function which returns either 1 (true) or 0 (false) depending on whether or not the command was executed on an Atari QL-Emulator. Unfortunately, there are some additional keywords only available on the Emulator, so a portable program which uses these has to check which system it is running on first.

**NOTE**

This function does not always work!

**CROSS-REFERENCE**

Also see *QDOS$*, *ATARI*, *VER$*, *QRAM$*, *WMAN$*, *P_ENV*. *MACHINE* is much more reliable.

## 24.35 QUB_DRIVE

| | |
|---|---|
| Syntax | QUB_DRIVE drive, card, file |
| Location | SMSQ/E for Q68 |

This command mounts a the first partition of a file formated like a Qubide drive. The file is specified by the card it is on and the filename

**Example**

```
QUB 1,1,"ql_bdi0.bin"
```

**Note 1**

Do not attempt to make two QUB drives point to the same partition!

**Note 2**

By default the QUB driver may not be loaded in SMSQ/E

**CROSS-REFERENCE**

*QUB_DRIVE$* returns the parameters already associated with a QUB drive.

## 24.36 QUB_DRIVE$

| | |
|---|---|
| Syntax | QUB_DRIVE$ (drive) |
| Location | SMSQ/E for Q68 |

This function returns a string containing the card and filename for specified QUB drive.

If the specified drive has not been linked to any particular file partition, an empty string is returned.

**Example**

```
QUB_DRIVE 1,1,"ql_bdi0.bin"
PRINT QUB_DRIVE$(1): REMark Will print QL_BDI0.BIN,1
```

**Note 1**

By default the QUB driver may not be loaded in SMSQ/E

**CROSS-REFERENCE**

See *QUB_DRIVE*.

## 24.37 QUB_USE

| | |
|---|---|
| Syntax | QUB_USE [device] |
| Location | SMSQ/E for Q68 |

As with FLP_USE this allows you to assign another three letter description to the QUB device driver, so that it can be accessed by programs which do not allow you to alter their devices. If no device is specified, then the device name is returned to the default qub.

**Example**

```
QUB_USE mdv
```

will ensure that any further attempt to access mdv1_ will actually access qub1_. If you later use the command:

```
QUB_USE
```

or:

```
QUB_USE qub
```

then you will once again be able to use the microdrives as well as qub1_.

**Note 1**

The QL's operating system tests for directory device drivers in a fixed order: DEV, FLP, RAM, WIN and MDV. This means that if you rename a driver to three letters which refer to a device driver earlier in the list, that original device driver will be used in preference. For example:

```
QUB_USE flp
```

will not work (attempts to read a file from flp1_ will still try to read floppy disk drive number one) - you will need to also rename the floppy disk driver:

```
FLP_USE flp
```

**Note 2**

By default the QUB driver may not be loaded in SMSQ/E

**CROSS-REFERENCE**

*FLP_USE*, *RAM_USE*, *DEV_USE*, *WIN_USE* are similar. *DMEDIUM_TYPE* can be used to find out the type of device which a name actually refers to. *DMEDIUM_NAME$* will return the default name of a device.

## 24.38 QUB_WP

| | |
|---|---|
| Syntax | QUB_WP drive, protect |
| Location | SMSQ/E for Q68 |

This command allows you to mark a specified QUB drive as read only protect=1 will write protect the hard disk. protect=0 (the default) will remove the write protection.

**Note 1**

By default the QUB driver may not be loaded in SMSQ/E

**CROSS-REFERENCE**

*DMEDIUM_RDONLY* will tell you if a device is read only.

## 24.39 QUEUE%

| | |
|---|---|
| Syntax | QUEUE% (string$) |
| Location | QBASE (DIY Toolkit Vol Q) |

QUEUE% is a function but does exactly the same as FORCE_TYPE and TYPE_IN.

The return value is zero if all bytes have been successfully typed in, negative if the keyboard queue is full and positive if another problem occurred.

The absolute value of the return always indicates how many characters QUEUE% failed to send.

## 24.40 QUIT

| | |
|---|---|
| Syntax | QUIT [return code] |
| Location | SMS |

This command is used to force remove a Multiple SBASIC Interpreter or a compiled Job (in the latter case it is the same as STOP).

An optional error code may be supplied to pass back to the calling job. This is only effective if the calling job is waiting for this one - ie, started with *EXEC_W*, *FEW* etc, otherwise the returned error code is ignored.

**NOTE**

If this command is used from SuperBASIC Job 0, it will return an 'Incomplete' error.

**CROSS-REFERENCE**

See *MB*, *FEW* and *SBASIC*. *CLOSE* #0 has the same effect from within a Multiple SBASIC or Multi-BASIC Interpreter

CHAPTER

# TWENTYFIVE

# KEYWORDS R

## 25.1 RAD

| | |
|---|---|
| Syntax | RAD (angle) |
| Location | QL ROM |

This function is used to convert an angle in degrees into an angle in radians (which is the system used by QDOS to represent angles in commands such as SIN, COS, TAN, etc.). Although this will work for any value of angle, due to the very nature of angles, angle should be in the range 0..360, which will return a value in the range 0..2Pi.

**Example**

A small program to draw a circle split into 30 degree segments:

```
100 MODE 4:WINDOW 448,200,32,16:SCALE 100,0,0
110 STRIP 2:INK 7
120 x1=75:y1=50:x2=x1:y2=y1+25
130 CIRCLE x1,y1,25
140 FOR x=0 TO 360/30-1
150   x2=x1+SIN(RAD(30)*x)*25:y2=y1+COS(RAD(30)*x)*25
160   LINE x1,y1 TO x2,y2
170 END FOR x
```

**CROSS-REFERENCE**

See *DEG*, *SIN*, *COS*. Also please refer to the Mathematics section of the Appendix.

## 25.2 RAE

| | |
|---|---|
| Syntax | RAE (i,n) |
| Location | Toolfin |

The function RAE returns the value of: i/(((1+i)$^{1/n}$) -1) *n ) where i and n can be any floating point numbers (see MT for error handling).

**CROSS-REFERENCE**

*MT*, *VA*, *VFR*, *VAR*, *TCA*, *TNC*, *TEE*, *RAFE*

## 25.3 RAFE

| | |
|---|---|
| Syntax | RAFE (i,n) |
| Location | Toolfin |

The function RAFE returns the value of $(((1+i)^{(1/n-1)} * n)/i$ where i and n can be any floating point numbers (see MT for error handling).

**CROSS-REFERENCE**

*MT*, *VA*, *VFR*, *VAR*, *TCA*, *TNC*, *TEE*, *RAE*

## 25.4 RAMTOP

| | |
|---|---|
| Syntax | RAMTOP |
| Location | Beuletools |

The function returns the active memory in KBytes, whether this is free memory or not. On the original QL without any expansion this function will return 128, whereas on a QL expanded to 640K, 640 is returned.

**CROSS-REFERENCE**

The value of *RAMTOP* can be lowered with *RESET* to emulate a machine which has less available memory.

*PRINT* (PHYSTOP/1024)-128 gives the value of the additional memory.

*FREE_MEM* and *FREE* return the actually available memory.

## 25.5 RAM_USE

| | |
|---|---|
| Syntax | RAM_USE [device] |
| Location | Trump Card, Gold Card, ST/QL, SMS |

This command is the same as FLP_USE except that it alters the name of the device used to access the ramdisks.

**NOTE**

device must only be three letters long.

**CROSS-REFERENCE**

See *FLP_USE*.

## 25.6 RAND

| | |
|---|---|
| Syntax | RAND (devicename) or<br>RAND (device$) |
| Location | TinyToolkit |

Every physical directory device (eg. floppies and microdrives, but not RAM disks), contain a random number in their FAT (which is a place on the medium which contains internal data, the number of free/bad/empty sectors etc). The function RAND returns this number which can be used by QDOS to check if a medium has been changed.

**NOTE**

QDOS increases access speed by temporarily storing data in so-called slave blocks. On QDOS and compatible machines this may lead to the phenomenon that RAND only returns the correct value the first time that it is used, and then returns the same value for any other medium. This can be avoided by using the command DEL_DEFB, or better CLRMDV before each RAND.

**CROSS-REFERENCE**

*FOP_DIR* opens a directory. *FLP_SEC* allows you to alter the amount of checking carried out by a disk interface to see if a disk has been swapped.

## 25.7 RANDOMISE

| | |
|---|---|
| Syntax | RANDOMISE [(start)] or<br>RANDOMISE \ (Minerva v1.82+ only) |
| Location | QL ROM |

SuperBASIC uses a standard method to generate pseudo random numbers.

Each time that the function RND is called, a start value specified by the command RANDOMISE is multiplied by a large number and another number is added, the solution is modulated and the final number is nearly unpredictable.

This method means that after a RANDOMISE command with parameter, RND will always generate the same numbers. If you pick another number as the parameter (or do not specify a parameter at all), this will let RND produce other numbers. If no parameter is specified, RANDOMISE calculates a random number by combining some internal system data such as the time, date, pointers etc.

**Example**

```
100 RANDOMISE 38
110 FOR n=1 TO 10: PRINT RND(10)
```

The program will always write the same ten random numbers on the screen.

**MINERVA NOTE**

The main problem with using RANDOMISE is that even without a parameter, the same pattern of 'random' numbers is generated if the interval between when the computer is switched on and when the line containing RANDOMISE is reached tends to be the same every time that a program is run. Although v2.53+ of SMS makes RANDOMISE without a parameter more random, the only other fix is on v1.82+ of Minerva which allows you to use:

```
RANDOMISE \
```

instead of the normal RANDOMISE which should alleviate this problem somewhat.

**CROSS-REFERENCE**

*RND* is the function that returns a random number. *RANDOMISE* influences *MATRND* in the same way.

## 25.8 READ

| | |
|---|---|
| Syntax | READ var[1] *[var[i]]* |
| Location | QL ROM |

This command forces the interpreter to look at the current data pointer, that is to say the next program line which begins with the marker DATA.

When a program is *first* RUN, the data pointer is set to the start of the program, and hence READ will attempt to assign the first value following the first occurrence of DATA to the specified variable. Having assigned this value, the data pointer is updated to the next value in the same DATA statement, or the next program line if there are no more values following the DATA statement.

If no more DATA is contained within the program and you try to READ a value, the error 'End of File' is reported. SMS's improved interpreter reports 'End of DATA'.

**Example**

A simple program to convert the three letter code returned by DAY$ into the full name of the day:

```
100 DATA 'Monday','Tuesday','Wednesday','Thursday'
110 DATA 'Friday','Saturday','Sunday'
120 RESTORE
130 dday$=DAY$
135 day=(dday$ INSTR ' MonTueWedThuFriSatSun')/3
140 FOR find_day=1 TO day
150   IF EOF:EXIT find_day
160   READ dday$
170 END FOR find_day
180 PRINT dday$
```

**NOTE 1**

The data pointer is not reset to the start of the program following every RUN. You need a RESTORE command to do this - try running the above program without line 120.

**NOTE 2**

If you try to READ a value into a slice of an undimensioned string, the value will not be stored and SuperBASIC may stop without a message, eg:

```
100 DATA 'Bess '
110 a$='Hello World'
120 READ a$(7 TO)
130 PRINT a$
```

Try by way of comparison, adding the line:

```
10 DIM a$(12)
```

Both of these work under SMS.

**NOTE 3**

The interpreter does not really check the parameters listed after READ, and unacceptable parameters, for example:

```
READ 1
```

won't report an error and the program continues as if it had read a variable. SMS's improved Interpreter reports 'Unacceptable Parameters for READ' when the program is RUN.

**MINERVA NOTE**

As from v1.96, READ has been improved so that it will accept an array parameter and then read a value for each element of the array in turn, without having to put READ into a loop.

**Example**

```
100 DIM x(5)
110 READ x
120 :
1000 DATA 1,2,3,4,5,6
```

This will read x(0), x(1), x(2), x(3), x(4) and x(5)

All other implementations insist upon you using something akin to:

```
100 DIM x(5)
110 FOR i=0 TO 5:READ x(i)
120 :
1000 DATA 1,2,3,4,5,6
```

**CROSS-REFERENCE**

*RESTORE* allows you to alter the program line pointed at by the data pointer. *DATA* sets out lines to be *READ*. *EOF* allows you to test for the end of all program data.

## 25.9 READ_HEADER

| | |
|---|---|
| Syntax | error = READ_HEADER(#channel, buffer) |
| Location | DJToolkit 1.16 |

The file that is opened on the given channel has its header data read into memory starting at the given address (buffer). The buffer address must have been reserved using *RESERVE_HEAP*, or some similar command.

The buffer must be at least 64 bytes long or unpredictable results will occur. The function will read the header but any memory beyond the end of the buffer will be overwritten if the buffer is too short. After a successful call to this function, the contents of the buffer will be as follows :

| Address | Value | Size |
|---|---|---|
| Buffer + 0 | File length | 4 bytes long (see *FILE_LENGTH*) |
| Buffer + 4 | File access | 1 byte long - currently zero |
| Buffer + 5 | File type | 1 byte long (see *FILE_TYPE*) |
| Buffer + 6 | File dataspace | 4 bytes long (see *FILE_DATASPACE*) |
| Buffer + 10 | Unused | 4 bytes long |
| Buffer + 14 | Name length | 2 bytes long, size of filename |
| Buffer + 16 | Filename | 36 bytes long |

Directory devices also have the following additional data :

| Address | Value | Size |
|---|---|---|
| Buffer + 52 | Update date | 4 bytes long (see *FILE_UPDATE*) |
| Buffer + 56 | Reference date | 4 bytes long - see below |
| Buffer + 60 | Backup date | 4 bytes long (see *FILE_BACKUP*) |

Miracle Systems hard disc's users and level 2 users will find the files version number stored as the the 2 bytes starting at buffer + 56, the remaining 2 bytes of the reference date seem to be hex 094A or decimal 2378 which has no apparent meaning, this of course may change at some point!

This function returns an error code if something went wrong while attempting to read the file header or zero if everything went ok. It can be used as a more efficient method of finding out the details for a particular file rather than calling all the various FILE_XXX functions. Each of these functions internally call the READ_HEADER routine.

To extract data, use *PEEK* for byte values, *PEEK_W* for the filename length and version number (if level 2 drivers are present, see LEVEL2), or *PEEK_L* to extract 4 byte data items.

The filename can be extracted from the buffer by something like:

```
f$ = PEEK_STRING(buffer + 16, PEEK_W(buffer + 14)).
```

**EXAMPLE** The following example allows you to change the current dataspace requirements for an *EXEC*utable file:

```
6445 DEFine PROCedure ALTER_DATASPACE
6450   LOCal base, loop, f$, ft, nv
6455   base = RESERVE_HEAP (64)
6460   IF base < 0 THEN
6465     PRINT "ERROR: " & base & ", reserving heap space."
6470     RETurn
6475   END IF
6480   REPeat loop
6485     INPUT'Enter filename:';f$
6490     IF f$ = '' THEN EXIT loop
6495     ft = FILE_TYPE(f$)
6500     IF ft < 0 THEN
6465       PRINT "ERROR: " & ft & ", reading file type for " & f$ & "."
6510     END IF
```

(continues on next page)

(continued from previous page)

```
6515     IF ft <> 1 THEN
6520       PRINT f$ & 'is not an executable file!'
6525       NEXT loop
6530     END IF
6535     PRINT 'Current dataspace is:'; FILE_DATASPACE(f$)
6540     INPUT 'Enter new value:'; nv
6545     OPEN #3,f$ : fer = READ_HEADER (#3,base)
6550     IF fer < 0 : CLOSE #3 : PRINT "READ_HEADER error: " & fer : NEXT loop
6555     POKE_L base + 6,nv
6560     fer = SET_HEADER(#3,base)
6565     IF fer < 0 : PRINT "SET_HEADER error: " & fer
6570     CLOSE #3
6575   END REPeat loop
6580   RELEASE_HEAP base
6585 END DEFine ALTER_DATASPACE
```

**CROSS-REFERENCE**

*SET_HEADER*, *FILE_LENGTH*, *FILE_TYPE*, *FILE_DATASPACE*, *FILE_UPDATE*, *FILE_BACKUP*.

## 25.10 RECHP

| | |
|---|---|
| Syntax | RECHP address or<br>RECHP address[1] *[,address[i]]* (BTool only) |
| Location | Toolkit II, THOR XVI, BTool |

The common heap is an area in memory where all programs may store data, this space being only limited by the memory available. A BASIC program can reserve space in the common heap with the function ALCHP.

The command RECHP allows you to recover this memory. The parameter of RECHP must be the address which was returned by ALCHP. The Btool variant of this command allows you to recover several addresses at once.

**Example**

Loading a title screen:

```
100 Title$="FLP1_TITLE_SCR"
110 IF FREE_MEM < 38*1024 THEN
120   LBYTES Title$,SCREEN
130 ELSE
140   TitleAdr=ALCHP(32768)
150   LBYTES Title$,TitleAdr
160   SCRBASE TitleAdr: REFRESH
170   RECHP TitleAdr
180 END IF
```

**NOTE**

RECHP reports error -15 if the address was not reserved with ALCHP or if the memory has already been given back to QDOS.

**CROSS-REFERENCE**

*CLCHP* clears all memory reserved by *ALCHP*, *CLEAR* deletes the values of all variables. See also *DISCARD*, *TTREL* and *RELEASE*.

## 25.11 RECOL

| | |
|---|---|
| Syntax | RECOL [#ch,] black,blue,red,magenta,green,cyan,yellow,white (MODE 8) or<br>RECOL [#ch,] black,1,red,3,green,5,white,white (MODE 4) |
| Location | QL ROM |

This command recolours all individual pixels in the specified window (default #1).

At least eight parameters must be specified, representing each of the colours available in MODE 8.

Each parameter must then have a value in the range 0..8 representing how that colour pixel is to be recoloured.

The rather odd syntax for use in MODE 4 is due to a slight apparent bug in the RECOL command which means that on some implementations the parameter which would normally represent the colour to replace yellow on screen has to be used to specify the colour to replace white.

**Example**

A simple demonstration program which recolours a circle randomnly:

```
100 WINDOW 448,200,32,16
110 PAPER 0:CLS:INK 7
120 SCALE 100,0,0
130 REPeat loop
140   CIRCLE 75,50,20
150   new_col=RND(1 TO 6)
160   RECOL 0,1,2,3,4,5,6,new_col
170 END REPeat loop
```

Note how this only works in MODE 8 (except on SMS): to get it to work in MODE 4, you would need to alter line 160 to:

```
160 RECOL 0,1,2,3,4,5,new_col,new_col
```

**NOTE 1**

Do not forget that the value of each parameter is taken to be the new colour, therefore if RECOL is to have no effect at all, you will need to use:

```
RECOL 0,1,2,3,4,5,6,7
```

and not:

```
RECOL 0,0,0,0,0,0,0,0
```

as you might at first think (the latter turns the whole window to black!).

**NOTE 2**

This command did not work on ST/QL Emulators prior to Level D-05 drivers.

**CROSS-REFERENCE**

*INK*, *FILL* See also *W_SWOP*, *SET_RED* and *SET_GREEN*.

## 25.12 REFRESH

| | |
|---|---|
| Syntax | REFRESH |
| Location | Fast PLOT/DRAW Toolkit |

This command forces the whole screen pointed to by SCRBASE to be copied onto the visible part of memory.

**NOTE**

REFRESH assumes 512x256 pixel resolution, the screen base is always assumed at $20000.

**CROSS-REFERENCE**

See also *SCRBASE*, *SCLR*, *PLOT* and *DRAW*. See also *W_SHOW*.

## 25.13 RELEASE

| | |
|---|---|
| Syntax | RELEASE address |
| Location | TinyToolkit |

This command allows you to return a section of memory reserved by GRAB to QDOS.

**NOTE**

LOAD, CLEAR, NEW and similar commands do not free GRABbed memory (unlike memory reserved with ALCHP).

**WARNING**

Never free memory where extensions, device drivers or other code have been loaded and started (for example with CALL) because the operating system will continue to update these routines regularly and find code which may have been overwritten by other programs, internal data etc. Crash!

**CROSS-REFERENCE**

*RECHP* and *CLCHP* clear memory allocated with *ALCHP*. *DISCARD* releases memory allocated with *RESERVE*. See also the other version of *RELEASE*.

## 25.14 RELEASE

| | |
|---|---|
| Syntax | RELEASE nr |
| Location | ST/QL, QSound |

RELEASE activates the enhanced sound capabilities of the ST/QL (or the QSound interface which has now been out of production for some years). A sequence which has been previously stored with PLAY under the number nr is 'executed' by RELEASE.

**CROSS-REFERENCE**

*PLAY*, *SND_EXT* Beware the other version of *RELEASE*.

## 25.15 RELEASE_HEAP

| | |
|---|---|
| Syntax | RELEASE_HEAP address |
| Location | DJToolkit 1.16 |

The address given is assumed to be the address of a chunk of common heap as allocated earlier in the program by *RESERVE_HEAP*. In order to avoid crashing the QL when an invalid address is given, RELEASE_HEAP checks first that there is a flag at address-4 and if so, clears the flag and returns the memory back to the system. If the flag is not there, or if the area has already been released, then a bad parameter error will occur.

It is more efficient to RELEASE_HEAP in the opposite order to that in which it was reserved and will help to avoid heap fragmentation.

**CROSS-REFERENCE**

See *RESERVE_HEAP*, below, for an example of use.

## 25.16 RELEASE_TASK

| | |
|---|---|
| Syntax | RELEASE_TASK jobnr, jobtag |
| Location | TASKCMDS (DIY Toolkit Vol J) |

See REL_JOB and RELJOB below. Refer to NXJOB for information about the job identification.

## 25.17 RELJOB

| | |
|---|---|
| Syntax | RELJOB jobId |
| Location | BTool |

Same as REL_JOB apart from the fact that this expects the JobID of the Job rather than its name or a simple job number.

## 25.18 RELOAD

| | |
|---|---|
| Syntax | RELOAD program_name |
| Location | MutiBASIC (DIY Toolkit - Vol M) |

This command is the opposite to UNLOAD in that it fetches the program which is stored in memory and loads it into the current SuperBASIC interpreter. If the screen mode has been stored with UNLOAD (or RESAVE), then when the program is loaded, RELOAD checks if the current display mode is the correct one and if not will alter it (although see below).

**NOTE 1**

See the various notes and warnings given for UNLOAD.

**NOTE 2**

Any commands which appear after RELOAD will be ignored.

**NOTE 3**

If you RELOAD a program which has a stored screen in a different mode to the current display mode, then the system can become confused if the Pointer Environment or Speedscreen is loaded. Therefore you should always ensure that the correct MODE is set before you RELOAD a program.

**NOTE 4**

If the specified file is not a file you stored with UNLOAD or does not exist, an error will be generated. You may also get the error 'Channel not Open' if the program uses a channel which was OPEN when the program was UNLOADed but is no longer OPEN.

**CROSS-REFERENCE**

*SCR_SAVE* allows you to dictate whether the screen display and mode should be stored together with the program. *REMOVE* allows you to remove a program stored in memory with this command. See also *RESAVE* and *QLOAD*.

## 25.19 REL_JOB

| | |
|---|---|
| Syntax | REL_JOB jobname or REL_JOB jobnr |
| Location | TinyToolkit |

This command releases a suspended job, so that it becomes active again.

**NOTE 1**

Releasing a job which is waiting for screen input/output will normally kill it, because it should be activated by <CTRL><C>.

**NOTE 2**

Before v1.11 of this Toolkit, jobnr could not be a variable (see JBASE).

**CROSS-REFERENCE**

Jobs can be suspended by *SJOB* and removed with *RJOB*, *KJOB*, *KILL*, etc. *JOBS* lists the current jobs. See *RELJOB*.

## 25.20 REMAINDER

| | |
|---|---|
| Syntax | REMAINDER |
| Location | QL ROM |

This keyword can only be used within a SELect ON structure. It is used to represent all possible untested values of the SELect ON variable.

**CROSS-REFERENCE**

Please see *SELect ON*.

## 25.21 REMark

| | |
|---|---|
| Syntax | REMark text |
| Location | QL ROM |

This command has no purpose when a program is RUNing. It is however used to place comments in the program which can be useful when you later come to edit a SuperBASIC program. Anything which appears after REMark on the same line, will be ignored by the interpreter, thus allowing you to make any sort of comment you like.

**Example**

> 100 REMark Line 110 could be altered to: 101 REMark 110 INPUT 'Your name';a$:IF password$<>a$:STOP 110 Name$='Author'

**CROSS-REFERENCE**

Another means of splitting a SuperBASIC program into sections is to include program lines which only contain a colon (:), for example:

```
100 PRINT "End of Program":STOP
110 :
200 DATA 'Some data to read'
```

## 25.22 REMOVE

| | |
|---|---|
| Syntax | REMOVE program_name |
| Location | MultiBASIC (DIY Toolkit - Vol M) |

This command allows you to remove a task (or program stored in memory with UNLOAD or RESAVE) by reference to its name. It is therefore very similar to RJOB, REL_JOB and REMOVE_TASK (amongst others).

## 25.23 REMOVE_TASK

| | |
|---|---|
| Syntax | REMOVE_TASK jobnr, jobtag |
| Location | TASKCMDS (DIY Toolkit - Vol J) |

Please see RJOB, because REMOVE_TASK a,b works like RJOB a,b,0.

## 25.24 RENAME

| | |
|---|---|
| Syntax | RENAME [device_]oldname TO [device_]newname |
| Location | THOR XVI, Toolkit II |

This command allows you to alter the name of a file which has already been created on the given device.

You must first of all specify the name of the file to be renamed (if no device is specified, the default data directory will be used). You will then need to specify the new name for that file (again, if no device is specified, the default data device will be used). Assuming that both filenames are valid, an attempt will be made to alter the filename as requested. If however newname already exists an error will be generated.

**Example**

```
RENAME flp1_boot TO flp1_oldboot
```

**NOTE 1**

If you try to RENAME a file across to another drive, (eg:

```
RENAME flp1_boot, flp2_oldboot
```

the error 'bad name' will be reported.

**NOTE 2**

Although you can RENAME each file within a sub-directory so that they no longer appear in that sub-directory, any attempt to RENAME the sub-directory itself (even if there are no files in it) will cause the error 'Read Only'.

For example, assuming that a directory of disk flp1_ returns the following:

```
boot QUILL->
```

You could for example, use:

```
RENAME flp1_QUILL_boot TO flp1_ARCHIVE_boot
```

if you wished, but any attempt to use:

```
RENAME flp1_QUILL TO flp1_ARCHIVE
```

will cause an error except on SMSQ/E (although an error is still generated on RAM disks).

**NOTE 3**

Unless you have Minerva v1.77 (or later) fitted, RENAME will alter the date of a microdrive file when used to rename a file on microdrive.

**NOTE 4**

In versions of Toolkit II before v2.10, RENAME could leave the file open (and therefore inaccessible) if only one name was provided.

**NOTE 5**

If you try to use RENAME to change a filename to upper case (or lower case) the error 'Already Exists' will be reported.

**CROSS-REFERENCE**

See also *WREN* which allows you to rename several files at once. *TTRENAME* is similar.

## 25.25 RENUM

| | |
|---|---|
| Syntax | RENUM [start_line [TO end_line];][new_line][,step] or<br>RENUM [start_line] TO [end_line][;new_line][,step] |
| Location | QL ROM |

When developing a SuperBASIC program, you will find that you sometimes run out of space in which to insert a new line, because of the line numbers which you have used. Line numbers can be any integer in the range 1…32767 and it is therefore unlikely that you will not be able to make room to fit any more lines into the program. To make more room, you will need to RENUMber the program. You can either elect to use RENUM in its simplest form, or a more complex form.

The simplest form of RENUM is the command:

```
RENUM
```

This will renumber the whole of the SuperBASIC program in memory, so that the first line number becomes line 100 and every subsequent SuperBASIC line number will be in an increment of 10.

You can however also use RENUM to renumber a specified range of lines in a program, by using some of the optional parameters. These parameters have the following effects:

- Start_line specifies the first line to be RENUMbered (default 1).
- End_line specifies the last line in the range to be RENUMbered (default 32767).
- New_line the line number which the start_line will be RENUMbered to (default 100).
- Step specifies the gap between each new line number (default 10).

RENUM will also alter line numbers referred to in the standard QL ROM commands:

```
GO SUB
GO TO
RESTORE
```

provided of course that the line number referred to is within the range of lines being renumbered!.

If the line number originally referred to does not exist, then RENUM will point it to the next program line following that line number.

It is also possible that a reference to a line number is actually calculated when the interpreter reaches that line. In such instances, the line number reference can only be renumbered if it is the first thing in the expression. For example, take the following program:

```
100 locat = 0
110 REPeat loop
120   RESTORE locat + 1000
125   IF EOF: EXIT loop
130   READ description$
140   PRINT description$
150   locat = locat + 1
155   PAUSE
160 END REPeat loop
888 :
1000 DATA 'Location One'
1001 DATA 'Location Two'
1002 DATA 'Location Three'
```

RENUM would renumber all of the line numbers beginning with line 100 in steps of 10, however, the program would no longer work as the RESTORE command in line 120 would then point to a non-existant line 1000. To solve this, before using RENUM, alter line 120 to:

```
120 RESTORE 1000 + locat
```

Having carried out the renumbering task, if the lines currently shown in the list window are affected, they will be relisted in #2 (except under SMS).

**Examples**

```
RENUM 100
```

or:

```
RENUM 1 TO
```

These are both the same as RENUM.

```
RENUM 100 TO 1000;10,5
```

This will renumber all lines in the range 100 to 1000, with the new lines beginning from line 10 in steps of 5.

```
RENUM 1000;2000
```

This will renumber all lines from line 1000 onwards, with the new line numbers beginning with line 2000, and increasing in steps of 10.

```
RENUM 1000,20
```

This will renumber all lines from 1000 onwards, with the new line numbers beginning with line 100 and increasing in steps of 20.

**NOTE 1**

On pre Minerva v1.77 ROMs, RENUM will not generally work correctly on the line number reference in a RESTORE where this appears on the same line as a DATA statement.

**NOTE 2**

On non Minerva ROMs, the current DATA pointer and ERLIN line numbers tend to get lost in the process! Although SMS updates the DATA pointer, it still has some problems. For example, try the following program:

```
1 RENUM TO 170;1,1
2 RESTORE
3 READ x:PRINT x
4 RENUM
5 READ x:PRINT x
6 RESTORE 6: DATA 10,12: RESTORE 6
7 READ x: PRINT x
8 STOP
180 PRINT 'Why have I reached here?'
```

If you alter line 1 to read:

```
1 RENUM 1,1
```

then the program just stops without an error at line 4. Minerva still has problems with the above.

Try entering the command:

```
RENUM 1 TO 7;1,1
```

An out of range error is reported even though there is no problem with this range. Minerva does this correctly. We believe other ROMs will show different symptoms (see the WARNING below).

**NOTE 3**

On Minerva ROMs (pre v1.97), if integer tokenisation is enabled, RENUM cannot renumber line numbers less than 128.

**NOTE 4**

You cannot use RENUM to renumber lines out of sequence. For example, given the following lines:

```
100 REPeat loop
110 IF INKEY$=CHR$(27):EXIT loop
120 END REPeat loop
```

Any attempt to:

```
RENUM 110 TO 110;200
```

would report an 'Out of Range' error, as you would be trying to renumber line 110 out of order!

**NOTE 5**

If you try to renumber a line outside of the range of line numbers (see above), or there is not enough space between line numbers outside the given range to fit the newly renumbered program lines into, this will cause an 'Out of Range' error. For example, taking the routine listed at note 4:

```
RENUM 32760
```

or:

```
RENUM 100 TO 110;119,1
```

would both report such an error.

**NOTE 6**

The Turbo and Supercharge compilers from Digital Precision cannot compile a program with calculated RESTOREs, GO SUBs or GO TOs.

**NOTE 7**

Unfortunately, RENUM will not handle line number references in commands other than GO TO, GO SUB or RESTORE, which can leave lines such as:

```
SAVE flp1_Prog_ext,1000 TO 2000
```

high and dry!

**WARNING**

It is generally inadvisable to use RENUM within a program as the interpreter tends to lose its place (see Note 2 above).

**CROSS-REFERENCE**

*DLINE* allows you to delete lines from a program. *ED* allows you to edit a program in memory. Also see *AUTO*.

## 25.26 REPeat

| | |
|---|---|
| Syntax | REPeat identifier or<br>REPeat [identifier](SMS only) |
| Location | QL ROM |

The SuperBASIC REPeat loop is extremely flexible and provides an alternative to the classic FOR loop.

It sets up a perpetual loop which can only be ended (correctly) by means of the EXIT command. The syntax of this SuperBASIC structure can take two forms:

REPeat identifier :statement *[:statement]*

or:

REPeat identifier *[statements]* ... [EXIT identifier] [NEXT identifier] ... END REPeat identifier

The first of these variants is known as an in-line REPeat loop. Provided that there is at least one statement following REPeat, this line will be repeated forever (unless there is an EXIT statement - see below). There is no need for a related END REPeat statement and therefore the shortest (practicable) in-line REPeat loop possible is:

```
REPeat loop: IF INKEY$=' ' THEN EXIT loop
```

If an in-line loop is terminated with EXIT, control will be passed to the statement following the corresponding END REPeat statement (if one exists), or the next program line. This allows the following:

```
REPeat loop: IF INKEY$=' ':EXIT loop: END REPeat loop: PRINT 'Phew!'
```

EXIT is used (in both REPeat loops and FOR loops) to terminate the loop, and the next statement which will be processed is the first statement after the corresponding END REPeat (if one exists).

NEXT forces the program to make another pass of the loop, returning program control to the statement following REPeat.

**Example**

A short FuNction which waits for a key to be pressed which can be <ESC> or any key listed in a string passed as the parameter, and returns the CODE of the key pressed:

```
100 DEFine FuNction Getkey(key$)
105   LOCal loop,k$
110   REPeat loop
120     k$=INKEY$:IF k$='':NEXT loop
130     IF k$ INSTR key$&CHR$(27):RETurn CODE(k$)
140   END REPeat loop
150 END DEFine
```

**NOTE 1**

The loop identifier must be a floating-point, except under Minerva or SMS. However, if the loop identifier is also used as a variable in the program, its value will not be altered by the REPeat / END REPeat / EXIT / NEXT statements. It can therefore still be used as a variable within the loop without any problems.

**NOTE 2**

It is actually possible to force a NEXT loop from outside of the loop, for example in a program such as:

```
100 REPeat Getkey
110   AT 0,0:PRINT 'Looping'
120   a$=INKEY$:IF a$='':NEXT Getkey
130   PRINT a$
140   IF a$=='x':EXIT Getkey
150 END REPeat Getkey
155 :
160 PRINT 'You have decided to leave the loop'
170 PRINT 'Press a key to return to it'
180 PAUSE
190 CLS
195 :
200 NEXT Getkey
```

This is however very bad programming style and should be avoided. It makes it very difficult to follow programs and no SuperBASIC compilers would be able to make sense of it. The above program should be re-written:

```
100 REPeat Getkey
110   AT 0,0:PRINT 'Looping'
```

(continues on next page)

(continued from previous page)

```
120   a$=INKEY$:IF a$='':NEXT Getkey
130   PRINT a$
140   IF a$=='x'
150     PRINT 'You are now still in the loop'
160     PRINT 'Press a key to restart it'
170     PAUSE
180     CLS
190   END IF
200 END REPeat Getkey
```

**MINERVA NOTES**

This allows string REPeat loops and integer REPeat loops, although the use of the former is dubious. You can of course still use the identifiers within the loop as variables. Integer REPeat loops do not seem to be any quicker than floating point loops.

If you do use a string identifier, Minerva restricts such strings to a maximum of four characters. If the string identifier is defined as a variable beforehand, it will be truncated if necessary - for example, try:

```
a$='Hello World': REPeat a$: PRINT a$ and a$='': REPeat a$: a$ = a$ & 'x':␣
↪PRINT a$
```

String and integer REPeat loops will not safely work on other ROMs (except under SMS), even if they will let you type them in!

**SMS NOTES**

Like Minerva, SMS allows string REPeat loops and integer REPeat loops. However, SMS does not restrict the length of a string loop identifier (except to the normal string length limit of 32767 characters). SMS also allows you to omit the loop identifier, in which case the relative EXIT, NEXT and END REPeat statements must also omit the loop identifier. This flexibility brings this command more in line with other implementations of BASIC. Error trapping of incorrectly structured REPeat loops is also improved - please refer to NEXT and END REPeat.

**CROSS-REFERENCE**

*FOR…END FOR* is the other loop type.

## 25.27 REPLACE

| | |
|---|---|
| Syntax | REPLACE oldvar, newvar |
| Location | REPLACE (DIY Toolkit - Vol R) |

The REPLACE command is intended for use from the interpreter's command line and for program development only.

The idea of REPLACE is to rename SuperBASIC variables contained in the program which is currently loaded into the interpreter. The first and second parameter can be any variables, they must not be given as strings ie. inside quotes (this leads to error -15: bad parameter).

REPLACE will replace oldvar by newvar for the whole program (in fact for the whole interpreter).

Acceptable types of parameters are variables and also REPeat loop names but not PROCedure or FuNction names.

You can even use this to change unquoted device names if you wish, such as:

```
LBYTES flp1_data_cde
```

You could use:

```
REPLACE flp1_data_cde, flp2_data_cde
```

REPLACE is extremely fast, without any noticeable reduction in speed for large programs due to the fact that the interpreter stores the program lines in tokenised format, this means that a line is not stored as text but as a set of numbers (tokens) which represent the elements of the line. So REPLACE merely has to modify the name table and change the name which is identified with a certain token.

**Example**

Enter the following lines:

```
10 x = 1
20 PRINT SQRT(x)
```

Now type:

```
REPLACE x, Whatever
```

and then LIST or ED, the program now reads:

```
10 Whatever = 1
20 PRINT SQRT(Whatever)
```

and is functionally identical to the original.

**NOTE 1**

Never use REPLACE as part of a program.

**NOTE 2**

REPLACE will work on a program loaded into a MultiBASIC.

**WARNING 1**

There is one possibility that you can harm your program: if you replace a variable by another variable which is already used in this program then the program will usually behave very differently after the REPLACEment.

**WARNING 2**

According to the Minerva Technical Manual REPLACE is "not particularly safe". At least if you are using the original version as published in QL World then you need to turn off Minerva's integer tokenisation (POKE \\212,128). Later versions (v0.3+) do however cope with integer tokenisation. Despite the warning, we have yet to find any other problems with REPLACE.

**CROSS-REFERENCE**

*NEW_NAME* is very similar to *REPLACE* but the parameters are passed as strings. This has the advantage that *NEW_NAME* can take variable parameters, *REPLACE* would replace the variable for the variable name. Compare *ALIAS*.

## 25.28 REPLY

| | |
|---|---|
| Syntax | REPLY [([#wind,] keys$)] |
| Location | BTool |

The function REPLY reads a character from the keyboard (with the text cursor in a window enabled).

If keys$ was specified, then REPLY will only stop if the pressed key was listed in keys$, this is case-sensitive so <a> and <SHIFT><A> are different.

The return of REPLY is the position of the pressed key in keys$. REPLY behaves very differently if there is no keys$ supplied. The return will be the code of the pressed key, just like CODE(INKEY$(-1)) except that combinations of <ALT> and any other key are recognised - if <ALT> was held and any other key pressed, REPLY returns 256 minus the code of that key.

**Example**

Another version of the game also shown at ASK:

```
100 CLS: x1 = 0: x2 = 100
110 PRINT "I am going to find out a number"
120 PRINT "from"!x1!"to"!x2!"which only you know."
130 PRINT "Press <S> if the proposed number is too small,"
140 PRINT "<L> if it's too large or <Y> if it's the result."
150 REPeat find_out
160   x=(x1+x2) DIV 2
170   PRINT x;"? ";
180   answer = REPLY("sSlLyY")
190   SELect ON answer
200     =1,2: x1 = x + 1: PRINT "too small"
210     =3,4: x2 = x - 1: PRINT "too large"
220     =5,6: EXIT find_out
230   END SELect
240 END REPeat find_out
250 PRINT "ok"\"I am the best."
```

**CROSS-REFERENCE**

*ASK*, *INKEY$* See *CODE* also.

## 25.29 REPORT

| | |
|---|---|
| Syntax | REPORT [#channel] or<br>REPORT [#channel,][error_number](Toolkit II, THOR XVI, TinyToolkit, BTool) |
| Location | QL ROM (post JM), Toolkit II, TinyToolkit, THOR XVI and BTool |

This command will print an error message to the given channel (default #0, the command line). The type of error is identified by the error number. If an error number is not supplied, then the last error to have occurred is displayed. The error message depends on the machine where the program is running, see ERNUM for conventions. Positive error numbers have no effect.

**WARNING**

Toolkit II's REPORT allows any value for the error_number, whereas TinyToolkit and BTool limit them to -1 to -21 and report undefined error for values lower than -21. Except under SMS, with Toolkit II, negative errors smaller than -27 may lead to undefined actions ie. printing a continuous stream of characters to the report channel - this may never stop.

**NOTE 1**

TRA can be used to redefine the error messages.

**NOTE 2**

For the original REPORT (QL ROM), only the first version of the command can be used. Further, if the supplied channel is not yet open, no error is reported and REPORT simply returns to BASIC as if it had carried out its job successfully. Also, on Minerva, SMS and ST/QL Emulators with E-Init software v1.27+, REPORT will show the line and statement number where the error occurred (rather than merely the line number) in the form: At line <line number>;<statement number><error message>

**CROSS-REFERENCE**

See *ERNUM* about error messages in general and *TK2_EXT* /*TINY_EXT* about updating Toolkits. Refer to the Appendix for the different message texts in various languages.

## 25.30 RESAVE

| | |
|---|---|
| Syntax | RESAVE program_name |
| Location | MutiBASIC (DIY Toolkit - Vol M) |

This command is the same as UNLOAD except that if the specified program_name has already been stored in memory, it is overwritten.

**CROSS-REFERENCE**

See *UNLOAD*!

## 25.31 RESERVE

| | |
|---|---|
| Syntax | RESERVE (bytes, JobID) |
| Location | Timing Toolkit (DIY Toolkit Vol H) |

This function grabs an area of memory in the Common Heap similar to ALCHP. However, the area is not released after a new SuperBASIC program is loaded. Standard error returns are returned as values by the function and the program can therefore include error trapping. -3 (Out of Memory) or -2 (Invalid Job ID) are the most common errors. You can also specify a task which will own the memory, and that memory will be removed when that task is removed. This task will normally be 0 (SuperBASIC) or -1 (the current job).

**CROSS-REFERENCE**

See *DISCARD* and *LINKUP*. Also see *ALCHP*, *RESPR* and *GRAB*.

## 25.32 RESERVE_HEAP

| | |
|---|---|
| Syntax | buffer = RESERVE_HEAP(length) |
| Location | DJToolkit 1.16 |

This function obtains a chunk of memory for your program to use, the starting address is returned as the result of the call. Note that the function will ask for 4 bytes more than you require, these are used to store a flag so that calls to *READ_HEADER* do not crash the system by attempting to deallocate invalid areas of memory. If you call this function, the returned address is the first byte that your program can use.

**EXAMPLE**

The following example shows how this function can be used to reserve a buffer for *READ_HEADER*, described elsewhere.

```
1000 buffer = RESERVE_HEAP(64)
1010 IF buffer < 0
1020    PRINT 'ERROR allocating buffer, ' & buffer
1030    STOP
1040 END IF
1050 error = READ_HEADER(#3, buffer)

.....do something with buffer contents here

2040 REMark Finished with buffer
2050 RELEASE_HEAP buffer
```

**CROSS-REFERENCE**

*RELEASE_HEAP*, *ALCHP*, *RECHP*, *ALLOCATION*.

## 25.33 RESET

| | |
|---|---|
| Syntax | RESET [new_ramtop](Not SMSQ/E) or<br>RESET(SMSQ/E only) |
| Location | TinyToolkit, Beuletools, BTool, SMSQ/E, RES |

This command performs a system reset. Except under SMSQ/E, this can be used to simulate a system with less memory or to get old games and problem software running, you can reduce the available memory (via new_ramtop) to anything between 128K (TinyToolkit: 64K) and RAMTOP in 64K steps (RES and BTool set a maximum of 640K).

**NOTE**

Do not include this command in a program without asking the user to confirm that it is OK since the computer may be writing some essential data to disk at the time (or still have some in memory).

**CROSS-REFERENCE**

On Gold Cards use *RES_128* and *RES_SIZE* for a faster reset. Minerva allows you to use *CALL* 390,x to reset the system.

## 25.34 RESFAST

| | |
|---|---|
| Syntax | RESFAST (bytes) |
| Location | ATARI_REXT for QVME (v2.31+) |

This function allows you to grab a specified number of bytes in Atari TT FastRAM and is therefore akin to RESPR and ALCHP. However, note that you can only use LBYTES to load data to this area or SBYTES / SEXEC to save data if you are loading a file from or saving a file to a RAM disk. You cannot use floppy disks or hard disks with this area of memory.

**CROSS-REFERENCE**

See FAST_FREE and *RESPR*.

## 25.35 RESPR

| | |
|---|---|
| Syntax | RESPR (bytes) |
| Location | QL ROM |

This function sets aside a chunk of resident procedure space for use by a program and returns the address of the start of that memory. Resident procedure space is merely an area of RAM which can be used safely by the user without fear of the system crashing if values are written to it.

When used, the RESPR function will search for an area in RAM which is currently unused and which is at least 'bytes' long. If there is insufficient space in RAM, then an 'Out of Memory' error is reported.

Memory set aside using RESPR cannot later be released and used for other purposes (unless you have a Minerva ROM), and thus this command is used mainly for linking in Toolkits and other system extensions in a boot program.

**Example**

A simple boot program might look like this:

```
100 x=RESPR(10*1024): LBYTES flp1_Toolkit,x: CALL x
120 EXEC flp1_Program_obj
```

**NOTE 1**

If a task is running in memory (eg. with EXEC), when RESPR is used, the resident procedure space cannot be accessed and the error 'Not Complete' is reported. However, some Toolkits, SMS and Minerva rewrite the RESPR command so that it will access the common heap if the resident procedure space cannot be accessed.

**NOTE 2**

Normally, the function RESPR(0) will return the address of ramtop, this can actually be used to find out the size of memory attached to the QL:

```
PRINT RESPR(0)/1024-128.
```

However, this will not work on versions of the command which work when tasks are running in memory.

**NOTE 3**

On Minerva pre v1.96, adding machine code functions and procedures from within a SuperBASIC PROCedure or FuNction definition could cause problems after a CLEAR command.

**WARNING**

Several programs may try to use the same area of resident procedure space if absolute addresses are used.

**CROSS-REFERENCE**

Please also see *ALCHP* which allocates memory from the common heap, which can be accessed when tasks are running in memory. Also see *RESERVE* and *GRAB* which are similar to *ALCHP*. It is also worth looking at *RESFAST*.

## 25.36 RESTORE

| | |
|---|---|
| Syntax | RESTORE [line_no] |
| Location | QL ROM |

In any program which uses DATA statements, it is necessary to tell the interpreter where the data begins within the program, so that it knows where to look when it encounters a READ command. RESTORE allows you to set the data pointer to a specific line number within a SuperBASIC program.

If line_no is not specified, then the data pointer is moved to the start of a program allowing all DATA within a program to be READ. line_no can be either a simple reference to a line number anywhere in a SuperBASIC program, or an expression which will be calculated by the interpreter when it reaches the RESTORE command.

**NOTE 1**

The Turbo and Supercharge compilers cannot compile computed RESTOREs.

**NOTE 2**

The data pointer is not reset when a program is RUN and it is therefore necessary to use an implicit RESTORE or CLEAR if you wish to read the same set of DATA each time that a program is RUN.

**NOTE 3**

On some implementations RESTORE with an invalid parameter will do a RESTORE 0. This is fixed on Minerva v1.96+ and SMS which report the error.

**CROSS-REFERENCE**

See *DATA* and *READ*. Please also refer to *RENUM*.

## 25.37 RES_SIZE

| | |
|---|---|
| Syntax | RES_SIZE ram_top |
| Location | Gold Card |

To get the few old programs which still do not work with the Gold Card's 1920K RAM running and to simulate a system with less RAM for debugging, RES_SIZE resets the system and adjusts the RAMTOP to the desired value.

If you use RES_SIZE 128, high density and extra density disks cannot be accessed until the next reset. Secondly, the realtime clock runs by default in protected mode. Thirdly, the ramdisks cannot be accessed by the system. This should simulate the unexpanded, original QL. Normal disk drives (DD) can still be accessed, although this can be temperamental.

**Examples**

```
RES_SIZE 640
RES_SIZE 128
RES_SIZE 1024
```

**NOTE**

You may find that some programs will still not work following RES_SIZE, especially if they use a line such as:

```
x=RESPR(0): start=RESPR(x-131072)
```

This appears to happen because RESPR(0) returns the address of RAMTOP which is still over 2MB even though only 128K is available. Minerva users should use:

```
CALL 390,x
```

instead.

**WARNING**

At least up to Gold Card's firmware v2.28, RES_SIZE does not check the range of the supplied parameter. If values lower than 56 or higher than 1920 are used, this can lead to crashes of a particularly serious character. Either the QL hangs during or after the resets, or there will not be enough free memory to enter any commands.

There is even a danger that a fatal crash will occur which can destroy data on hard disks, disks or microdrive cartridges, or the realtime clock can be affected or even combinations of these different crashes can occur. As hard disk drives cannot be removed or protected from any malfunction of the operating system or programs, they are in extreme danger.

It is also not advisable to use values other than multiples of 64 because software tends to expect a ramtop which is a multiple of 64 and memory is wasted.

**CROSS-REFERENCE**

*RES_128* is identical to *RES_SIZE* 128. See also *RESET*. See *RAMTOP* and *FREE_MEM* about available and free memory. *FLP_EXT* improves the reliability of the floppy disk drives and allows RAM disks to be used.

## 25.38 RES_128

| | |
|---|---|
| Syntax | RES_128 |
| Location | Gold Card, Trump Card |

This command does the same as RES_SIZE 128.

**CROSS-REFERENCE**

*FLP_EXT* can be used to re-enable some functions such as ramdisks.

## 25.39 RETRY

| | |
|---|---|
| Syntax | RETRY or<br>RETRY [line_no](Toolkit II and Minerva) |
| Location | QL ROM, Toolkit II |

The command RETRY performs the same operation as CONTINUE except that interpreting re-starts with the statement at which the error occurred (CONTINUE re-starts the program from the next statement).

If you have Toolkit II or Minerva installed, you will be able to use the second variant of this command which allows you to re-start processing at a specified line number to help with error trapping. If the parameter is specified, this is exactly the same as the second variant of CONTINUE.

**Example**

Take the following short program:

```
100 REPeat loop
110 INPUT 'Enter a number: ';a
120 PRINT 'The number you entered is: ';a
130 END REPeat loop
```

Now, when prompted to enter a number, enter a letter, which results in the error 'Error in Expression'. If you were to enter the command RETRY, the program would re-start at line 110, asking you to enter a number. However, if you entered the command CONTINUE, the program would re-start at line 120, displaying the message:

```
The number you entered is: *
```

**CROSS-REFERENCE**

Please refer to *CONTINUE*!

## 25.40 RETurn

| | |
|---|---|
| Syntax | RETurn [expression] |
| Location | QL ROM |

This command has two actual uses. The main use of RETurn is to force an early return from a PROCedure or FuNction definition block. A FuNction must always return a value and therefore a SuperBASIC DEFine FuNction block must always contain a RETurn statement to return this value.

The second use of RETurn is to mark the end of a sub-routine which has been called with GO SUB. This is implemented in SuperBASIC to make the transition from other implementations of BASIC easier.

**Examples**

A PROCedure to report an error more safely than REPORT:

```
100 DEFine PROCedure REPORT_ERROR(errnumber)
110 IF errnumber>=0 OR errnumber<-21
120   PRINT #0,'No error'
130   RETurn
140 END IF
150 REPORT errnumber
160 END DEFine
```

A FuNction which returns 1 (true) if a given number is even:

```
100 DEFine FuNction CK_EVEN (x)
110   IF x/2=INT(x/2):RETurn 1
120   RETurn 0
130 END DEFine
```

**CROSS-REFERENCE**

See *DEFine PROCedure* and *DEFine FuNction*. Please also refer to *GO SUB*.

## 25.41 REV$

| | |
|---|---|
| Syntax | REV$ (string$) |
| Location | REV |

This function returns the supplied string in reverse order.

**Example**

```
PRINT REV$("Hello World")
```

shows dlroW olleH

**CROSS-REFERENCE**

*LEN* finds the length of a string. *TRIM$* cuts off excess spaces from a string.

## 25.42 RJOB

| | |
|---|---|
| Syntax | RJOB jobname [,error] or<br>RJOB jobnr,tag,error or<br>RJOB job_id,error or<br>RJOB [job_id,error] (BTool only) |
| Location | Toolkit II, THOR XVI, BTool |

This command removes a job from memory - all of its channels are automatically closed and any memory used by the job is freed. The error code is returned to the owner job of the removed job. The BTool variant of RJOB allows you to enter the command without any parameters which will kill every job except SuperBASIC (Job 0), see KJOBS and KILL.

**NOTE**

If the first syntax does not work, you are using an old Toolkit version.

**CROSS-REFERENCE**

*KJOB* works similarly to *RJOB*. *KILL*, *REMOVE* and *KJOBS* remove all jobs. Have a look at *JOBS*, *SPJOB*, *AJOB*, *SJOB* etc.

## 25.43 RMAR

| | |
|---|---|
| Syntax | RMAR(n) with n=0..255 |
| Location | Beuletools |

This function returns the control codes needed to set the right margin to n characters (from the left side) on EPSON compatible printers. If the right margin is smaller than the left margin, the printer will ignore this setting and print to the greatest possible right margin:

```
PRINT #ch,RMAR
```

is the same as:

```
PRINT #ch,CHR$(27) & 'Q' & CHR$(n)
```

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*, *DBL*, *ENL*, *PRO*, *SI*, *NRM*, *UNL*, *ALT*, *ESC*, *FF*, *LMAR*, *PAGDIS*, *PAGLEN*.

## 25.44 RMODE

| | |
|---|---|
| Syntax | RMODE [(screen)] |
| Location | Fn |

The function RMODE returns the current screen mode (of the screen belonging to the job which executes RMODE if the Window Manager is present).

If Minerva or Amiga QDOS v3.23 is present and is in dual screen mode, then PRINT RMODE(1) will show the current screen mode for the Other Screen (see MODE). If Minerva and Amiga QDOS is not present, (or dual screen mode is not active), then RMODE(1) will return -19 (for 'Not Implemented'). Both RMODE and RMODE(0) return the mode of the Default Screen on all ROMs:

| RMODE | Min Resolution | Colours |
|---|---|---|
| 2 | 640 x 400 | 2 |
| 4 | 512 x 256 | 4 |
| 8 | 256 x 256 | 8 |
| 12 | 256 x 256 | 16 |

**Example**

If a program is written to operate in one of these modes, it has to change to that mode at the very beginning. A simple MODE 4 will do, if high resolution is needed. But the MODE is executed even if the screen was already in the correct mode. It looks better if MODE is only done if the mode really has to be changed. CHANGE_MODE should be used instead of MODE:

```
100 DEFine PROCedure CHANGE_MODE (mode%)
110   IF RMODE(0)<> mode%
120     MODE mode%
130   END IF
140 END DEFine CHANGE_MODE
```

**CROSS-REFERENCE**

*MODE* sets the mode. *QFLIM* returns the screen resolution. *TTMODE%* is similar.

## 25.45 RND

| | |
|---|---|
| Syntax | RND [([min TO] max)] |
| Location | QL ROM |

This function produces a (pseudo) random number. When used without parameters it returns a floating point number between 0 and 1, otherwise an integer number lying between the two parameters (including the parameters) will be returned.

| Expression | Results |
|---|---|
| x=RND | 0 < x < 1 |
| x=RND(max) where max >= 0 | 0, 1, 2, 3, . . . , max |
| x=RND(min TO max) where max >= min | min, min+1, . . . , max-1, max |

**Example**

```
100 CLS: PRINT "RND Statistics"
110 n = 1000: m = 10: DIM h%(m)
120 FOR i=1 TO n
130   k = RND(1 TO m)
140   h%(k) = h%(k )+ 1
150   AT 2,5: PRINT i
160 END FOR i
170 PRINT: avdiff = 0
180 FOR k = 1 TO m
190   diff = n / m - h%(k)
200   PRINT k; TO 6; h%(k); TO 12; INT(diff)
210   avdiff = avdiff + ABS(diff / n * m)
220 END FOR k
230 PRINT\"average difference:" ! INT(100 * avdiff / m);"%"
```

**NOTE**

If a range is specified {eg. RND(x TO y)} the second number must not be less than the first (ie. y>=x). If only one parameter is specified, this is taken to be the top of the range, with the bottom of the range

being 0. Therefore, if only one parameter is specified, this must not be negative.

**CROSS-REFERENCE**

The results of *RND* can be influenced with *RANDOMISE*. See also *MATRND*.

## 25.46 ROM

| | |
|---|---|
| Syntax | ROM (n) |
| Location | TinyToolkit |

This function returns the address in memory where additional ROMs can be placed. The parameter specifies the number of the slot you wish to look at (it must be in the range 0…16). The possible values are:

| n | ROM(n) |
|---|---|
| 0 | 49152 (EPROM-Port) |
| 1 | 786432 |
| 2 | 802816 |
| 3 | 819200 |
| 4 | 835584 |
| 5 | 851968 |
| 6 | 868352 |
| 7 | 884736 |
| 8 | 901120 |
| 9 | 917504 |
| 10 | 933888 |
| 11 | 950272 |
| 12 | 966656 |
| 13 | 983040 |
| 14 | 999424 |
| 15 | 1015808 |
| 16 | 1032192 |

**CROSS-REFERENCE**

ROM_TEST checks if a piece of code can be placed into a ROM. *EPROM_LOAD* allows you to load an EPROM on an emulator.

## 25.47 ROM_EXT

| | |
|---|---|
| Syntax | ROM_EXT |
| Location | ATARI_REXT |

This command activates any EPROMs in a standard QL format which have been plugged into the ROM port on the Atari ST. The code contained in the EPROMs is initialised just as it would be on the QL.

**NOTE**

This can only be used on code which is stored on EPROM chips, as a QL EPROM cartridge cannot be plugged into the Atari ST.

**CROSS-REFERENCE**

See also *ROM_LOAD* and *EPROM_LOAD* which allows you to transport code across from QL EPROM cartridges.

## 25.48 ROM_LOAD

| | |
|---|---|
| Syntax | ROM_LOAD device_file |
| Location | ATARI_REXT (pre v1.21 only) |

On later versions of the Emulator, this has been renamed EPROM_LOAD.

## 25.49 ROMs

| | |
|---|---|
| Syntax | ROMs [#ch] |
| Location | Beuletools |

This command lists all ROM headers of plugged in ROMs to the given channel (default #1), provided the ROMs conform to the Sinclair standard. This will recognise, for example, Trumpcard, Atari QL-Emulator and anything plugged into the QL's ROMport.

**CROSS-REFERENCE**

*ROM* returns the start address of a ROM slot.

## 25.50 RTP_R

| | |
|---|---|
| Syntax | RTP_R (imag, real) |
| Location | PTRRTP |

The function RTP_R takes a given rectangular co-ordinate and returns the so-called module (ie. the radius in polar co-ordinates). The result of RTP_R is always strictly positive and is not affected by the sign of the imag and real parameters, because of the symmetries of a circle.

**Example 1**

Draw a rectangular pattern in green and the corresponding polar pattern again displayed as rectangular co-ordinates in white:

```
100 SCALE 10,-5,-5: PAPER 0: CLS
110 FOR x = -3 TO 3 STEP .4
120   FOR y = -3 TO 3 STEP 5E-2
130     INK 4: POINT x, y
140     INK 7: POINT RTP_R(x,y), RTP_T(x,y)
150   END FOR y
160 END FOR x
```

**Example 2**

The same as the above example but the polar co-ordinates are treated even more unusually. If you correct the program and exchange a and b in line 140 then the two patterns will match exactly - this reveals what the RTP_... functions are actually doing:

```
100 SCALE 10,-5,-5: PAPER 0: CLS
110   FOR x = -3 TO 3 STEP .4
120   FOR y = -3 TO 3 STEP 2E-2
130     INK 4: POINT x, y
140     a = RTP_R(x,y): b = RTP_T(x,y)
145     INK 7: POINT b * COS(a), b * SIN(a)
150   END FOR y
160 END FOR x
```

**CROSS-REFERENCE**

Polar co-ordinates also need an angle, this is calculated with *RTP_T*. The *PTR_X* and *PTR_Y* pair of functions are complementary to *RTP_R* and *RTP_T*.

## 25.51 RTP_T

| | |
|---|---|
| Syntax | RTP_T (imag, real) |
| Location | PTRRTP |

The function RTP_T takes rectangular co-ordinates and returns the corresponding argument, (the angle used in polar co-ordinates) in radians. See RTP_R for further information.

## 25.52 RUN

| | |
|---|---|
| Syntax | RUN [line] |
| Location | QL ROM |

There is one command which can be found in any BASIC language:

```
RUN
```

Issuing RUN may actually be a little closer to the truth than you like to admit, but you should be happy with BASIC. Assembly language is much more terrifying, and if you have not yet reached that point of knowledge and understanding which it is most frustrating to reach... However:

```
RUN line
```

is identical to:

```
GOTO line
```

and:

```
RUN
```

without a parameter, could be replaced by GOTO 1.

Unlike some implementations of BASIC, the variables and the DATA pointer are not reset when you enter RUN.

**NOTE**

Jobs cannot be started with RUN but have to loaded and executed with EX, EXEC_W,… or a file manager/desktop. RUN will work okay from inside compiled jobs to enable them to re-start themselves.

**CROSS-REFERENCE**

See *GO TO* or even better, *REPeat* and *FOR* loops.

CHAPTER

TWENTYSIX

# KEYWORDS S

## 26.1 SAR

| | |
|---|---|
| Syntax | SAR file, array |
| Location | ARRAY |

The command SAR allows you to save a given array quickly (so that it can later be reloaded) as a whole to a specified file. The Toolkit II default data device for the file name is supported, although an existing file is not overwritten (use SARO) - the error 'Already exists' will be generated instead. array is stored in an internal coded but portable format, which makes it extremely fast to save and load arrays when compared to storage by writing and reading each individual element of an array.

The file format is quite simple, it is basically the same as the way in which SuperBASIC itself would store the array. The first four bytes of the stored array are the characters WLAF. SAR will identify dimensions and the type of array on its own and accordingly store it.

Sub-arrays are handled, but please note that, since stored data can only be reloaded into a readily dimensioned array (see LAR), it is important to remember the dimensions and type of the array before loading.

**Example**

Save and load an array:

```
100 DIM a%(1000)
110 SAR file$, a%
120 LAR file$, a%
```

**NOTE**

SAR's file format is not compatible with that used by Toolkit 3 (a commercial Toolkit which has nothing to do with the famous Toolkit II), produced with SARRAY.

**CROSS-REFERENCE**

*SARO* and *LAR*. *DIM* sets-up an array while *DIMN* and *NDIM* read the dimensions.

## 26.2 SARO

| | |
|---|---|
| Syntax | SARO file, array |
| Location | ARRAY |

SARO is almost the same as SAR except that it overwrites an existing file without reporting an error.

## 26.3 SAUTO

| | |
|---|---|
| Syntax | SAUTO seconds |
| Location | Ecran Manager |

This is yet another screen saver.... It is activated by specifying how many seconds the computer should wait for a key to be pressed before it turns the screen blank. Once the screen is blank, any key will display the screen again. Negative seconds deactivate this most useful of all computer utilities.

**Example**

```
SAUTO 180
```

blanks the display if no key is pressed for three minutes.

**NOTE 1**

If seconds = 0 then an annoying flashing screen results, so avoid it.

**NOTE 2**

See SSAVE.

**CROSS-REFERENCE**

*SCRON*, *SCROF*, *MODE*

## 26.4 SAVE

| | |
|---|---|
| Syntax | SAVE device_filename *[,range]* or<br>SAVE [device_] filename *[,range]*(Toolkit II only) or<br>SAVE(SMS only) |
| Location | QL ROM, Toolkit II |

If no line range is given, this command saves the whole of the currently loaded SuperBASIC program to the given directory device, under the given filename. However, a range of lines can be given (as with LIST), in which case only the given lines will be saved. If the filename already exists on that device, the error 'Already Exists' is reported, unless you have Toolkit II present, in which case, a prompt will be printed in #0 asking you whether it is okay to overwrite that file. If the device is already full, the 'Device Full' error is reported, however, the effects should the drive become full during the actual SAVE command, depends upon the implementation (see below).

The file is saved in pure ASCII format, which means that it can be COPYed to the screen or a printer (using COPY_N). The Toolkit II variant of the command will add the data default directory to the filename if it cannot find the given device, or no device is specified.

**Examples**

Save the whole of the current program to microdrive 1 with the filename BOOT:

```
SAVE mdv1_BOOT
```

Save the whole of the current program to the current data default directory with the filename prog_bas:

```
SAVE prog_bas
```

Save lines 1, 100 to 150 (inclusive) and 300 to the end of the program to the current data default directory with the filename cut_bas:

```
SAVE cut_bas,1,100 TO 150,300 TO
```

**NOTE 1**

SAVE can lead to incomplete files if the Break key is pressed or the device fills up during access, although Toolkit II (v2.13+) will report any file errors during output (other than the Break key being pressed), leaving the incomplete file on the device.

**NOTE 2**

Minerva (pre v1.80) deleted the file if SAVE was aborted for any reason.

**NOTE 3**

If you try to SAVE a file on top of a sub-directory name, Toolkit II will repeatedly ask if it is OK to overwrite that file until you answer <N> (for No).

**SMS NOTES**

The third variant of the command will allow you to SAVE the program in memory under the same filename as when LOAD or QLOAD was last used (with the _BAS suffix appended if necessary). If the original filename used when the program was LOADed ended in _SAV, then SAVE will alter this to be the _BAS suffix. This variant will also take the version number of the file when it was LOADed (or QLOADed) and then increase this by one.

If you SAVE a file on a disk, then use DELETE to remove that file, and then change the disk before issuing the SAVE command without a filename being specified, SMSQ/E fails to recognise that the disk has been swapped and repeatedly tries to write out the file using the old directory map. Further, if you enter SAVE without a parameter and no disk is in the drive - SMSQ/E asks if it is OK to overwrite the file!!

**CROSS-REFERENCE**

*LOAD* loads a saved file from the given device into memory. *SAVE_O* is another variant of this command. See also *QSAVE* for a different means of *SAVE*ing a SuperBASIC program.

## 26.5 SAVE_O

| | |
|---|---|
| Syntax | SAVE_O device_filename *[,range]*(THOR XVI) or<br>SAVE_O [device_] filename *[,range]*(Toolkit II) or<br>SAVE_O (SMS only) |
| Location | THOR XVI, Toolkit II |

This command operates in exactly the same manner as SAVE, except that the file is automatically overwritten if it already exists.

**NOTE**

This will not overwrite a sub-directory file and will create the same problem as SAVE.

**CROSS-REFERENCE**

See *SAVE*!

## 26.6 SAVEPIC

| | |
|---|---|
| Syntax | SAVEPIC file$ |
| Location | PICEXT |

This command saves the screen contents (from $20000) to the specified file, which has to be given as a string and must include the full filename. The file which will be created is 32K long.

```
SBYTES file$, SCREEN, 32768
```

does exactly the same.

**NOTE**

SAVEPIC makes the same assumptions and suffers from the same compatibility problems as LOADPIC.

**CROSS-REFERENCE**

*LOADPIC* displays the saved screen file.

## 26.7 SB_THING

| | |
|---|---|
| Syntax | SB_THING |
| Location | SMSQ |

This command is found in versions of SMSQ which do not have the Hotkey System II built in (most QXL versions of SMSQ). It is used to create the SBASIC Executable Thing so that you can start SBASIC up from a Hotkey or by using the EXEC set of commands.

**NOTE**

You must only use this command after the file HOT_REXT has been loaded, for example with LRESPR flp1_HOT_REXT.

**CROSS-REFERENCE**

Please refer to *EW* and *SBASIC*. Also see the Appendix on Multiple BASICs.

## 26.8 SBASIC

| | |
|---|---|
| Syntax | SBASIC [pos] or<br>SBASIC pos$ |
| Location | SMS |

This command is used to start up a Multiple SBASIC interpreter, which is nearly an exact copy of the main interpreter (Job 0) and will contain a copy of all of the toolkit commands used by the parent Job when this command is invoked. Any toolkits subsequently loaded into the new SBASIC interpreter cannot be used by its parent and vice versa. In its simplest form:

```
SBASIC
```

a new Interpreter will be started up which has windows #0,#1 and #2 open as per Job 0.

You can however pass either a one or two digit number (either as a numeric pos or a string pos$), in which case only #0 will be opened and its position will depend upon the number which has been passed as a parameter. This enables you to start up a new SBASIC Interpreter without its windows overlapping existing programs.

If only one digit is passed, this is taken to be the SBASIC row number. Row 0 is at the top of the screen, Row 1 is 64 pixels from the top, Row 2 128 pixels from the top and so on…

If two digits are passed, the first digit is taken to be the SBASIC column number, the second becomes the SBASIC row number (see above).

The column number is calculated as, Column 0 being the left hand side of the screen, Column 1 is 256 pixels from the left, Column 2 512 pixels from the left and so on….

**Example**

The following examples should make it clear:

```
1000 REMark Opens three channels, #0, #1 and #2 over the top of, and the
1005 REMark same size as the existing windows in SuperBASIC;
1010 SBASIC
1015 :
1020 REMark Opens one channel, #0, as con_256x62a0x128;
1025 SBASIC 2: REMark Or SBASIC '2'
1030 :
1035 REMark Opens one channel, #0, as con_256x62a0x192;
1040 SBASIC 3: REMark Or SBASIC '3'
1045 :
1050 REMark Opens one channel, #0, as con_256x62a256x64;
1055 SBASIC 11: REMark Or SBASIC '11'
1060 :
1065 REMark Opens one channel, #0, as con_256x62a256x128;
1070 SBASIC 12: REMark Or SBASIC '12'
1075 :
1080 REMark Opens one channel, #0, as con_256x62a768x128;
1085 SBASIC 32: REMark Or SBASIC '32'
```

**CROSS-REFERENCE**

See *MB* and *EW* for other ways of starting up additional interpreters. In addition SMS users can use *EXEP* SBASIC or even use the Exec button from QPAC II and Minerva users can use *EX* PIPEP. *WMON* and *WTV* can be used to reposition the SBASIC windows. Also see the appendix on Multiple BASICs. *JOB_NAME* can be used to alter the name of a SBASIC Job.

## 26.9 SBYTES

| | |
|---|---|
| Syntax | SBYTES device_file,start,length or<br>SBYTES device_file,start[,length[,data[,extra[,type]]] (Minerva v1.80+) or<br>SBYTES [device_]file,start,length(Toolkit II) or<br>SBYTES #channel,start,length(SMS only) |
| Location | QL ROM, Toolkit II |

It can sometimes be useful to save part of the QL's memory to a file so that it can be loaded back into the computer at a later date. The area of memory saved may for example, contain a program, some machine code or some data.

The command SBYTES allows you to save length number of bytes from the QL's memory, starting from the specified start address. The area of memory is saved to the specified file which must include the name of the device to be used, unless Toolkit II is present, in which case the default data device is supported. The Toolkit II variant will also provide you with the option of overwriting the file if it already exists.

**Example**

To save the currently displayed screen on a standard QL, use the command:

```
SBYTES flp1_Example_scr,SCREEN,32768
```

Or under SMS, you can save any size screen using:

```
SBYTES flp1_Example_scr, SCR_BASE, SCR_LLEN * SCR_YLIM
```

The start of a program which was protected by a password could be written along the lines of this:

```
100 a=ALCHP(100)
110 IF FTEST(flp1_pass)=0
120   LBYTES flp1_pass,a
130 END IF
140 pass$=''
150 PAPER#0,0:CLS#0:INK#0,7
160 PRINT #0,'Enter Password :';
170 FOR letter=1 TO 4
180   pass$=pass$&INKEY$(-1)
190   PRINT#0,'*';
200 END FOR letter
210 PRINT #0
220 IF PEEK(a)=0
230   offset=RND(1 TO 50)
240   POKE a,offset
250   FOR i=1 TO 4
260     POKE a+i,CODE(pass$(i))+i+offset
270   END FOR i
280   FOR i=5 TO 100:POKE a+i,RND(100)
290   SBYTES flp1_pass,a,100
```

(continues on next page)

(continued from previous page)

```
300 ELSE
310   offset=PEEK(a)
320   FOR i=1 TO 4
330     IF CODE(pass$(i))<>PEEK(a+i)-i-offset
340       PRINT 'Access Denied':RECHP a:STOP
350     END IF
360   END FOR i
370 END IF
380 PRINT 'Access Granted' 390 RECHP a
```

**NOTE 1**

On Minerva ROMs (pre v1.80), if SBYTES was aborted for some reason whilst writing to a file, the file would be deleted. On later versions of Minerva and all other QL ROMs, the incomplete file is kept and on Toolkit II, the error 'Medium Full' is reported.

**NOTE 2**

On Minerva ROMs (pre v1.83) SBYTES set the wrong file type.

**NOTE 3**

The Minerva variant is unfortunately overwritten by the Toolkit II version of this command.

**MINERVA NOTES**

On Minerva v1.80 (or later) the commands SBYTES and SEXEC have practically become interchangeable, as both support exactly the same parameters. All of the parameters except for the start address and device_file where the data is to be stored, are optional and will default to 0 if not specified.

These additional parameters have the following uses:

- Extra This sets the value which is normally returned with FXTRA (which would normally have to be altered with SetHEAD).
- Type This allows you to set two file attributes:
  - the file type - this is normally 0 for data, or 1 for executable programs. This is calculated by PRINT type && 255.
  - the file access key - it is generally used by Toolkits such as QL-System to store various file attributes (such as whether a file is read-only). This is calculated by PRINT type DIV 256.

The only problem with using this extended version of SBYTES instead of using SEXEC is that you must remember to specify a file type of 1 if the file is later to be EXECuted (as this defaults to 0 in the case of SBYTES!).

For example, both of these are the same:

```
SBYTES ram1_test_exe, code_start, 20000, 500, 0, 1
SEXEC ram1_test_exe, code_start, 20000, 500
```

**SMS NOTE**

The fourth variant of the command allows you to save the bytes to an existing channel which is already OPEN to a file, thus allowing you to work more efficiently. You can use the following to error trap the saving routine:

```
100 REPeat loop
110   ch=FOP_NEW (ram1_test_bin)
120   IF ch<0
130     REPORT ch
140     PRINT "Press <y> to retry, <n> to stop"
150     REPeat kLoop
160       key$=INKEY$(-1)
170       IF key$ INSTR 'yn':EXIT kLoop
180     END REPeat kLoop
190     IF key$=='y':NEXT loop
200     STOP
210   END IF
220   SBYTES #ch,131072,32768
230   EXIT loop
240 END REPeat loop
250 CLOSE #ch
```

**CROSS-REFERENCE**

*SBYTES_O* and *SEXEC* are very similar. *DATA_USE* allows you to alter the current default data device. *LBYTES* allows you to load in a block of code which has been saved with *SBYTES* or *SEXEC*.

## 26.10 SBYTES_O

| | |
|---|---|
| Syntax | SBYTES_O [device_]file,start,length(Toolkit II only) or<br>SBYTES_O device_file,start,length (THOR XVI) or<br>SBYTES_O #channel,start,length (SMS only) |
| Location | Toolkit II, THOR XVI |

This command is exactly the same as SBYTES except that it will automatically overwrite an existing file of the same name.

**NOTE**

The Toolkit II version of the command supports the default data device.

**CROSS-REFERENCE**

See *SBYTES*.

## 26.11 SCALE

| | |
|---|---|
| Syntax | SCALE [#ch,] size,x,y or<br>SCALE [#ch,] -size,x,y (Minerva v1.76+) |
| Location | QL ROM |

Many of the QL's graphics commands rely upon the graphics co-ordinate system to dictate whereabouts in a window they should appear. The command SCALE allows you to set the graphics scale in a specified

window (default #1).

Size dictates the graphics scale for that window by representing the length of a line which would be drawn from the bottom left hand corner of a window to the top left hand corner, hence the larger the size, the more information which can appear on screen (although this is somewhat limited by the actual resolution of the screen!). The default size is 100.

The co-ordinates x,y specify the co-ordinate which appears in the bottom left hand corner of the screen. When a window is opened, the scale is reset with the equivalent of *SCALE #ch,100,0,0*.

Although a line drawn up the side of a window will be size units long, the length required to draw a line along the whole of the bottom of the window, not only depends on the value of size, but also on the screen resolution and the shape of the given window.

```
size|
Y   |
    |
    |
    |
    |
    |__________________________________
    (0,0)        size*factor
                      X
```

**Example**

The following short program will draw a diagonal cross through the middle of any given size of window, on any given screen resolution:

```
100 INPUT 'Screen Resolution Width :'!ScreenX
110 INPUT 'Screen Resolution Height :'!ScreenY
120 INPUT 'Window Width : ';wid
130 INPUT 'Window Height : ';hi
140 INPUT 'Scale : ';size
150 WINDOW wid,hi,32,16
160 PAPER 2:INK 7:CLS
170 SCALE size,0,0
180 Xratio=ScreenX/512:Yratio=ScreenY/256
190 line_diff=(101*Yratio/hi)*wid/('135.5041505'*Xratio)
200 LINE 0,0 TO size*line_diff,size
210 LINE 0,size TO size*line_diff,0
```

**NOTE 1**

Graphics drawn using the QL graphics co-ordinate system will appear in the same place on screen in any screen MODE.

**NOTE 2**

Due to the QL's arithmetic routines, the maximum length of a line which can be drawn upwards in a window is slightly larger than size.

**NOTE 3**

On JSU ROMs, the screen ratio is different to other ROMs, presumably due to the different number of lines on American TVs - you would need to change line 190 in the above example to read:

```
90 line_diff=(101*Yratio/hi)*wid/('159.593001'*Xratio)
```

**MINERVA NOTES**

Minerva ROMs (v1.76 or later) allow you to use a negative SCALE, so that you may easily draw a picture upside down without altering all of the different drawing commands. This is achieved by using the second syntax of the SCALE command. Before trying to use this new variant of the command, you will have to give your drawing some careful thought.

For instance, when designing a screen, it is best to draw this using the normal SCALE command, and then to use Minerva's new syntax at that stage. For example, take the following short demonstration drawing:

```
100 MODE 8
110 WINDOW 448,200,32,16
120 SCALE 100,0,0:PAPER 0:CLS
130 INK 2:FILL 1
140 LINE 0,0 TO 0,10
150 LINE 0,10 TO 40,30 TO 60,27
160 LINE 60,27 TO 40,24 TO 10,0 TO 0,0
170 FILL 1
180 LINE 166,0 TO 166,10
190 LINE 166,10 TO 126,30 TO 106,27
200 LINE 106,27 TO 126,24 TO 156,0 TO 166,0
210 INK 4,3
220 FILL 1:CIRCLE 83,50,32:FILL 0
```

Now, to turn it upside down on Minerva, try changing line 120 to read:

```
120 SCALE -100,0,0:PAPER 0:CLS
```

If you now try running the program, you will find that your picture no longer appears!

This is because instead of Minerva moving the graphics origin to the top right hand corner of the window (as you might have expected), Minerva has in effect turned the graphics output around by 180 degrees about the graphics origin (ie. the bottom left hand corner of the window).

This means that to get your picture to re-appear on screen, you will need to redefine the graphics origin so that it has the same values as you would expect to be in the top right hand corner of the screen before it was turned around. Thus, the following diagram will explain the current layout of the screen:

```
-size|
 Y   |
     |
     |
     |
     |
     |____________________________
     (0,0)    -size*factor
                   -X
```

In other words, in the above example, you will need to alter line 120 to read:

```
120 SCALE -100,166,100:PAPER 0:CLS
```

The program will now display the picture upside down, having now altered the orientation of the display to the following diagram:

```
     X
     ______________________________________(0,0)
size|                                      |
    |                                      |
    |                                      |
    |                                      |
    |                                      |
    |                                      | Y
    |______________________________________|
    (166,100)           size*factor
```

**CROSS-REFERENCE**

*CIRCLE*, *ARC*, *LINE* and *POINT* all rely on the current *SCALE*.

## 26.12 SCLR

| | |
|---|---|
| Syntax | SCLR colour |
| Location | Fast PLOT/DRAW Toolkit |

This command causes the screen (pointed to by SCRBASE) to be cleared with the given colour. This colour ranges from 0 (black) to 7 (white). There is no default.

**Example**

Run this small program and watch how your screen seems to change size. The greater the difference in apparent size, the worse your monitor (there are more intelligent criteria as to the capability of your monitor, for example radiance):

```
100 MODE 4:SCRBASE
110 REPeat screen_test
120   FOR n=0,2,4,6: SCLR n
130 END REPeat screen_test
```

**NOTE**

See REFRESH !

**WARNING**

Never use SCLR without a parameter! The system will crash!

**CROSS-REFERENCE**

*PLOT*, *DRAW*, *REFRESH* and *SCRBASE* are the other commands connected with this one.

## 26.13 SCRBASE

| | |
|---|---|
| Syntax | SCRBASE [adress] |
| Location | Fast PLOT/DRAW Toolkit |

All commands belonging to the fast graphics Toolkit use the specified parameter set with this command as the base address for their operations: SCLR, PLOT, DRAW and REFRESH. This is intended to allow background drawing. The default address is the address of the visible screen, SCREEN.

**Example 1**

A simple demonstration and an animated version:

```
100 SCRBASE ALCHP(32768): SCLR 0
110 FOR t=0 TO 2*PI STEP PI/32
120   x1=188*SIN(t)+255: y1=127*COS(t)+127
130   x2=188*SIN(t+PI)+255: y2=127*SIN(t+PI)+127
140   DRAW x1,y1 TO x2,y2 ,7
150 END FOR t
160 REFRESH: CLCHP
```

```
100 Pics=INT((FREE_MEM-4096)/32768)
110 DIM base(Pics)
120 FOR c=2*PI/Pics TO 2*PI STEP 2*PI/Pics
130   base(c/2/PI*Pics)=ALCHP(32768)
140   SCRBASE base(c/2/PI*Pics): SCLR 0
150   FOR t=0 TO 2*PI STEP PI/4
160     x1=188*SIN(t)+255: y1=127*COS(t)+127
170     x2=188*SIN(t+c)+255: y2=127*SIN(t+c)+127
180     DRAW x1,y1 TO x2,y2 ,7
190   END FOR t
200 END FOR c
210 :
220 REPeat Animation
230 FOR c=1 TO Pics: SCRBASE base(c): REFRESH
240   IF KEYROW(1)=8 THEN EXIT Animation
250 END REPeat Animation
260 CLCHP
```

**Example 2**

Varying the base address by steps equal to the value of SCRINC (normally 128) simulates vertical scrolling. The first program views memory, the other one loads an uncompressed 32K screen and then 'scrolls it in'.

```
100 FOR A=0 TO 786432 STEP 128
110   SCRBASE A
120   REFRESH
130 END FOR A
```

The second program appears on the next page.

```
100 SCRFILE$="MDV1_SCREEN_SCR"
110 SCROLLSPEED=4 120 :
130 ADR=ALCHP(65536)
140 LBYTES SCRFILE$,ADR+32768
150 POKE$ ADR,FILL$(CHR$(0),32767): POKE SCREEN+32766,0
160 FOR A=ADR TO ADR+32768 STEP SCROLLSPEED*SCRINC
170   SCRBASE A
180   REFRESH
190 END FOR A
200 RECHP ADR
```

**CROSS-REFERENCE**

See *SCLR*, *PLOT*, *DRAW* and *REFRESH* for fast background drawing. *SCR_STORE* and *SCR_REFRESH* are ideal to create and display animated displays. Compare *SCR_BASE*!

## 26.14 SCREEN

| | |
|---|---|
| Syntax | SCREEN or<br>SCREEN [(#ch)] (FN Toolkit only) |
| Location | Beuletools, Fn |

The visible screen on a standard QL is actually 32K of memory. The start address of the screen is normally 131072, but can change on Minerva and higher resolution implementations of the QL, so the start address should be determined before accessing the screen directly. The function SCREEN returns that start address.

**NOTE**

The FN Toolkit version allows you to specify a channel - if the channel is specified, then the start address for the screen on which that channel is open is returned. This is mainly only of use to Minerva and Amiga QDOS users who can have windows open on either the Default Screen or the Other Screen (provided their dual screen mode is active). Thus SCREEN(#3) can be used to find the start address of the second screen if that is where #3 is located.

**CROSS-REFERENCE**

See *SCR_BASE*.

## 26.15 SCREEN_BASE

| | |
|---|---|
| Syntax | screen = SCREEN_BASE(#channel) |
| Location | DJToolkit 1.16 |

This function is handy for Minerva users, who have 2 screens to play with. The function returns the address of the start of the screen memory for the appropriate channel.

If the returned address is negative, consider it to be a QDOS error code. (-6 means channel not open & -15 means not a SCR_ or CON_ channel.)

SCREEN_BASE allows you to write programs that need not make guesses about the whereabouts of the screen memory, or assume that if *VER$* gives a certain result, that a Minerva ROM is being used, this may not always be the case. Regardless of the ROM in use, this function will always return the screen address for the given channel.

**EXAMPLE**

```
PRINT HEX$(SCREEN_BASE(#0), 24)
```

## 26.16 SCREEN_MODE

| | |
|---|---|
| Syntax | current_mode = SCREEN_MODE |
| Location | DJToolkit 1.16 |

This function can help in your programs where you need to be in a specific mode. If you call this function you can find out if a mode change needs to be made or not. As the *MODE* call changes the mode for every program running in the QL, use this function before setting the appropriate mode.

The value returned can be 4 or 8 for normal QLs, 2 for Atari ST/QL Extended mode 4 or any other value deemed appropriate by the hardware being used. Never assume that your programs will only be run on a QL!

**EXAMPLE**

```
1000 REMark Requires MODE 4 for best results so ...
1010 IF SCREEN_MODE <> 4
1020    MODE 4
1030 END IF
1040 :
1050 REMark Rest of program ....
```

**CROSS-REFERENCE**

*MODE*.

## 26.17 SCRINC

| | |
|---|---|
| Syntax | SCRINC [(#ch)] |
| Location | Fn |

The screen width is not fixed on QDOS computers, QL Emulators and future hardware expansions (graphic cards) offer different screen modes with different resolutions. The function SCRINC returns the screen width relating to the screen upon which the given channel (default #0) is located. The width is returned as the number of bytes needed to store a line of pixels.

The standard QL mode 4 and mode 8 always return 128 = 512/4.

However, it is not *always* true that the number of bytes required to store a line of pixels is equal to the number of pixels DIV 4 and you should therefore use this function or similar.

**Example**

See the second listing at the second example for SCRBASE.

**CROSS-REFERENCE**

*SCREEN* returns the start address of the screen. See also *SCR_LLEN*

## 26.18 SCROLL

| | |
|---|---|
| Syntax | SCROLL [#ch,] distance [,area] |
| Location | QL ROM |

This command allows you to move the contents of a given window (default #1) up or down by a specified number of pixels (distance).

A positive value for distance will move the contents of the window downwards, whereas a negative distance will move them upwards.

As the contents are moved, if they move outside of the limits of the window, they will be lost. The space left by the movement of the window's contents, will be filled with the current PAPER colour.

If you use the third parameter (area), you can specify that only part of the window is to be moved, by using the following values:

- 0 This is the default - move whole window.
- 1 Move the area above the text cursor line.
- 2 Move the area below the text cursor line.

If you wish to move other areas of a window, the easiest method is to open another window over that part of the window which you want to move, and then use SCROLL and/or PAN on that new window (see example below).

**Example**

A short demonstration routine of SCROLL and PAN:

```
100 MODE 4
110 WINDOW 440,200,32,16: PAPER 2: CLS
120 INK 7: CSIZE 3,1
130 AT 0,6: PRINT 'QL KEYWORD MANUAL'
140 OPEN #3,scr_448x200a32x16: PAPER#3,2
150 AT 5,6: PRINT 'QL KEYWORD MANUAL'
160 FOR i=1 TO 37
170   WINDOW #3,40,200,432,16
180   SCROLL #3,20
190   PAUSE 5
200   WINDOW #3,440,20,32,16
210   PAN #3,40
220   PAUSE 5
230   WINDOW #3,40,200,32,16
240   SCROLL #3,-20
250   PAUSE 5
260   WINDOW #3,440,20,32,196
```

(continues on next page)

(continued from previous page)

```
270   PAN #3,-40
280   PAUSE 5
290 END FOR i
300 CSIZE 0,0
```

**NOTE**

QL ROMs (other than v6.41 of THOR XVI, SMS and v1.63/v1.64 of Minerva) allow SCROLL to be used to access various direct TRAP #3 calls to the operating system (as with PAN and CLS).

The first parameter to be supplied represents the D1 parameter in machine code, whereas the second parameter represents D0. In any case, both parameters must be integers (ie. in the range -32768..32767).

Normally to find out number to give D0, take the routine's D0 value and subtract 24 (eg. IOG.DOT=48, 48-24=24). However, if the routine's value is 24 or less, subtract 24 and then add this negative value to 128.

Some useful routines which can be accessed are:

- SCROLL #3,0,121 moves the cursor to column 0 in #3 (IOW.SCOL, D0=$11)
- SCROLL 0,24 has the same effect as CLS 16, ie. it calls (IOG.DOT - D0=$30), which effectively carries out the command POINT 0,0.
- SCROLL x,17 sets the ink colour to x (IOW.SINK,D0=$29)
- SCROLL #3,n%,42 sets the file pointer in #3 to n% (IOF.POSA,D0=$42)
- SCROLL #3,n%,43 should move the file pointer in #3 on n% places (IOF.POSR,D0=$43)

Unfortunately, not all values for both parameters will work on all ROMs and this is a hit and miss way of programming the QL. Luckily, the wealth of Toolkits available should mean that there is a legal means of accessing these routines, using Toolkit keywords, including MTRAP and QTRAP.

**CROSS-REFERENCE**

*PAN* allows you to move the contents of a window sideways. *WINDOW* allows you to specify the area of the screen which a window covers. *IO_TRAP* allows you to access machine code routines directly. See also *QTRAP*, *BTRAP* and *MTRAP*.

## 26.19 SCROF

| | |
|---|---|
| Syntax | SCROF |
| Location | Ecran Manager |

This command forces the current screen to become invisible - the effect of SCROF lasts until the next task switch under the Pointer Environment or until one of the standard MODE commands (ie. not dealing with dual screen mode) or NEW are issued.

**Example**

```
SCROF
```

**NOTE**

See SSAVE.

**CROSS-REFERENCE**

*SCRON* switches the screen to visible.

## 26.20 SCRON

| | |
|---|---|
| Syntax | SCRON |
| Location | Ecran Manager |

The SCRON command makes the screen visible once again after it has been disabled with SCROF.

**Example**

```
SCRON
```

**NOTE**

See SSAVE.

**CROSS-REFERENCE**

*SCROF*.

## 26.21 SCR2DIS

| | |
|---|---|
| Syntax | SCR2DIS |
| Location | Super Gold Card |

Some programs make use of the QL's ability to support a second screen (on a standard QL this is normally stored at $28000 (hex) - it overwrites the system variables which are moved to another area in memory). You can therefore see why it is important never to make assumptions about the location of the screen or system variables in memory (use SCREEN or SYS_BASE instead).

Minerva extends this second screen even further, allowing you to operate the computer in two-screen mode, with programs being started up on one of two screens (thus allowing you to have completely different displays on each screen) see MODE. The main problem with this second screen is that it slows down the operation of the computer and therefore if you do not intend to use the second screen, you may wish to disable it.

You can disable the second screen with the command SCR2DIS - this setting will be stored in memory by the Gold Card and the second screen will henceforth always be disabled.

**WARNING**

Some programs (mainly games) will not work properly with the second screen disabled.

**CROSS-REFERENCE**

*SCR2EN* re-enables the second screen again.

## 26.22 SCR2EN

| | |
|---|---|
| Syntax | SCR2EN |
| Location | Super Gold Card |

This command is the complementary command to SCR2DIS - it enables the QL's second screen and is also memorised by the Super Gold Card so that the second screen will always be available for use by programs.

**NOTE**

In order to make proper use of the second screen, you will still need to startup Minerva in dual screen mode and use the appropriate MODE commands. Non-Minerva QLs can still use the second screen by using various machine code techniques.

**CROSS-REFERENCE**

See *SCR2DIS* for more information.

## 26.23 SCR_BASE

| | |
|---|---|
| Syntax | SCR_BASE [(#ch)] |
| Location | ATARI_REXT (v2.25+), SMSQ/E |

This function returns the base address of the screen linked to the specified channel (default #0), this is normally 131072 on standard QLs but can alter on other resolutions or if dual screen mode is supported. On machines which support higher resolutions, the screen base will only be at the standard address of 131072 if you configure the machine to start up in 512x256 and even here there is no guarantee - see the documentation for the particular QL resolution you are using.

**NOTE**

If the specified channel is not open then Invalid Channel ID will be reported. However, if no channel is specified and #0 is not open, then a special window will be opened for #0 on screen, which may destroy what is already on screen.

**CROSS-REFERENCE**

*SCREEN* is similar. See also *SCR_XLIM*, *SCR_YLIM* and *SCR_LLEN*. *A_OLDSCR* can help some older software to work. You can also use *PRINT CHAN_L%*(#1,50) instead of *SCR_BASE*.

## 26.24 SCR_LLEN

| | |
|---|---|
| Syntax | SCR_LLEN [(#ch)] |
| Location | ATARI_REXT (v2.25+), SMSQ/E |

This function returns the number of bytes required to hold one line of pixels on the current screen resolution attached to the specified channel (default #0). On a standard QL 512x256 resolution, this is normally 128 bytes but can alter on other resolutions.

**NOTE**

As with SCR_BASE, if the specified channel is not open then Invalid Channel ID will be reported. However, if no channel is specified and #0 is not open, then a special window will be opened for #0 on screen, which may destroy what is already on screen.

**WARNING**

You should never assume that the number of bytes required to store a line is the number of pixels DIV 4 - always use this function instead.

**CROSS-REFERENCE**

*SCRINC* is similar. See also *SCR_XLIM*, *SCR_YLIM* and *SCR_BASE*. On QL ROMs after JM, you can also use *PRINT CHAN_B%*(#1,104).

## 26.25 SCR_REFRESH

| | |
|---|---|
| Syntax | SCR_REFRESH address |
| Location | SuperWindow Toolkit |

This command copies a screen (or a portion of a screen) which has been saved in memory at the specified address using SCR_STORE back to the same position on the visible display screen.

**NOTE**

It is unknown at present whether these commands check for the start address of the screen and its dimensions and therefore they may not work on some higher resolutions. Unfortunately, we do not have access to the toolkit at present.

**CROSS-REFERENCE**

*SCR_STORE* stores a window or rectangle taken from the display. See also *W_SHOW* and *REFRESH*.

## 26.26 SCR_SAVE

| | |
|---|---|
| Syntax | SCR_SAVE flag |
| Location | MutiBASIC v4.0+ (DIY Toolkit - Vol M) |

This command is used to specify whether the current screen display and mode should be stored along with the program when the UNLOAD or RESAVE commands are used. The setting depends on the value of flag:

- 0 Do not store the screen display and mode.
- 1 (This is the default). Store the screen display and mode so that it is redisplayed when RELOAD is used.
- -1 This tells RELOAD to ignore the screen details (if any) stored with the program - use SCR_SAVE 1 if you want to see them.

**NOTE**

Beware that this toolkit only supports 512x256 resolution and expects the screen base to be at 131072.

**CROSS-REFERENCE**

*UNLOAD* contains more details about this toolkit.

## 26.27 SCR_SIZE

| | |
|---|---|
| Syntax | SCR_SIZE [(#channel)] or<br>SCR_SIZE (width_x,width_y [[,pos_x],pos_y] ) |
| Location | SuperWindow Toolkit |

This function will return the space in bytes, a window (default #1) or rectangle on the screen, needs to be stored with SCR_STORE. Windows are specified just by reference to their channel number, whereas rectangles by their width and height. Naturally, the size of any shape is independent from its position but the co-ordinates may be also added as parameters without invoking an error message - or influencing the result of SCR_SIZE.

**WARNING**

SCR_SIZE with a channel number will not work correctly if the Window Manager is present because of the different window definition blocks. Use either the second syntax or on a standard QL calculate the size yourself: size=8+width_x*width_y/4

**CROSS-REFERENCE**

*SCR_STORE* stores a part of the screen in RAM and *SCR_REFRESH* copies it back. See also *WMAN$*. *CHAN_W%* is much more flexible.

## 26.28 SCR_STORE

| | |
|---|---|
| Syntax | SCR_STORE [#channel,] address or<br>SCR_STORE width,height,x,y TO address |
| Location | SuperWindow Toolkit |

This command allows you to store a part of the screen at the given address in RAM. The section of the screen to be stored can be either a window channel number (default #1) or the dimensions of a rectangle. The amount of memory SCR_STORE needs is returned by SCR_SIZE. SCR_STORE needs eight bytes plus the actual amount of space taken up by the section of the screen. These four words (one word consists of two bytes) are kept at the start of the storage area and contain the size and position of the screen part as passed by the second syntax above. They can easily be read like this: width = PEEK_W (adress) height = PEEK_W (adress+2) x = PEEK_W (adress+4) y = PEEK_W (adress+6)

**Example**

The SCR_STORE and SCR_REFRESH commands are ideal tools to create and show animations. The actual speed of SCR_REFRESH is independent from the contents of the screen, so it does not matter how long it took to create the pictures… Enjoy it.

```
100 wx=70: wy=70: px=100: py=100
110 OPEN#3,"scr_" & wx & "x" & wy & "a" & px & "x" & py: CLS#3
120 size=SCR_SIZE(wx, wy): DIM adr(20)
130 bx=2: by=2: pmax=10
140 :
150 FOR p=1 TO pmax
160   adr(p)=ALCHP(size)
170   FOR x=0 TO wx-bx STEP bx
```

(continues on next page)

(continued from previous page)

```
180     a=2*SQRT(p)*x/wx-SQRT(p)
190     FOR y=0 TO wy-by STEP by
200       b=2*SQRT(p)*y/wy-SQRT(p)
210       z=((a*a+b*b)^^(a*b-b*b)) MOD 7
220       BLOCK#3,bx,by,x,y,z
230     END FOR y
240   END FOR x
250 SCR_STORE wx,wy,px,py TO adr(p)
260 END FOR p
270 :
280 REPeat Animation
290   FOR p=1 TO pmax: SCR_REFRESH adr(p)
300   FOR p=pmax-1 TO 2 STEP -1: SCR_REFRESH adr(p)
310   IF KEYROW(1)=8 THEN EXIT Animation
320 END REPeat Animation
330 CLCHP
```

**CROSS-REFERENCE**

See *SCR_REFRESH* and *SCR_SIZE*. See also *W_STORE* and *W_CRUNCH*. Use *ALCHP* to set aside some memory to hold the copy of the window. Use *RECHP* to remove that memory definition.

## 26.29 SCR_XLIM

| | |
|---|---|
| Syntax | SCR_XLIM [(#ch)] |
| Location | SMSQ/E |

This function is the same as QFLIM(#ch,0) except that the channel parameter is optional (it defaults to #0).

**NOTE**

As with SCR_BASE, if the specified channel is not open then Invalid Channel ID will be reported. However, if no channel is specified and #0 is not open, then a special window will be opened for #0 on screen, which may destroy what is already on screen.

**CROSS-REFERENCE**

*QFLIM* and *XLIM* are similar. See also *DISP_SIZE* and *SCR_YLIM*

## 26.30 SCR_YLIM

| | |
|---|---|
| Syntax | SCR_YLIM [(#ch)] |
| Location | SMSQ/E |

This function is the same as QFLIM(#ch,1) except that the channel parameter is optional (it defaults to #0).

**NOTE**

As with SCR_BASE, if the specified channel is not open then Invalid Channel ID will be reported. However, if no channel is specified and #0 is not open, then a special window will be opened for #0 on screen, which may destroy what is already on screen.

**CROSS-REFERENCE**

*QFLIM* and *YLIM* are similar. See also *SCR_XLIM*, *SCR_BASE* and *SCR_LLEN*.

## 26.31 SDATE

| | |
|---|---|
| Syntax | SDATE year,month,day,hours,minutes,seconds<br>SDATE year,month,day,hours,minutes (SMS v2.57+)<br>SDATE time (Minerva, SMS) or<br>SDATE TO time (THOR XVI) |
| Location | QL ROM |

The QL has an internal clock which contains the current date and time. Unfortunately, this clock is corrupted every time that the QL is switched on and off (and even in some cases when the QL is reset). This means that the clock has to be set manually every time that the system is re-booted. Because of this, various battery-backed clocks have appeared on the market which retain the time whilst the QL is turned off and then the QL clock is generally reset to the same time as the battery backed clock when it is switched back on.

This command allows you to set the internal QL clock to a specified date and time. Each parameter in the first syntax must be a numeric value.

The second syntax is similar to the first, but is only supported on later versions of SMS. This variant accepts just five parameters and assumes that the seconds is to be set to zero.

The third and fourth syntaxes allow you to set the time and date by the number of seconds since Midnight on 1st January 1961. This thus allows you to copy the date from one QL to another very simply over the Network:

```
100 temp_file$='n1_ram1_temp'
110 er=FOP_NEW(temp_file$)
120 IF er>0
130 CLOSE #er:SDATE TO FUPDT(\temp_file$)
140 DELETE temp_file$
150 END IF
```

**Example**

```
SDATE 1993,1,1,0,0,0
```

sets the internal clock to the start of 1993.

**NOTE 1**

This may also affect battery backed clocks - see their instructions. In particular on the THOR XVI the battery backed clock is automatically reset, whereas on earlier THORs the command SET_CLOCK was needed.

**NOTE 2**

Unfortunately, current versions of Minerva and SMS will not accept the THOR's syntax, nor vice versa.

**NOTE 3**

On the QXL, before v2.57 of SMS the time would not be set correctly if seconds=0 or seconds=1. The clock could still be wrong by 1 second until v2.73 which fixed this problem on MOST PCs.

**CROSS-REFERENCE**

*PROT_DATE* allows you to prevent *SDATE* from altering a battery backed clock. *ADATE* allows you to alter the time by a specified number of seconds. *DATE* lets you read the current date and time as a single figure. *DATE$* and *DAY$* return various details about the current date and time. These functions can also be used to find out details concerning a given date without having to use *SDATE* beforehand to change the system date. *A_SDATE* and *SET_CLOCK* alter the battery backed clocks on the ST/QL Emulator and THOR respectively.

## 26.32 SDP_DEV

| | |
|---|---|
| Syntax | SDP_DEV device |
| Location | Gold Card, Trump Card, SDUMP_REXT, ST/QL |

The command SDP_DEV allows you to dictate where output from the SDUMP device should be sent. Initially, all output is sent to ser, however you may wish to alter this. Under SMS, you will need to LRESPR SDUMP_REXT provided on the distribution disk.

**Example**

SDP_DEV n1_flp1_Dump will cause all future output from the SDUMP device to be sent to a file flp1_Dump on the machine with NetID=1 in the Network.

**CROSS-REFERENCE**

*SDUMP* allows you to send output to the specified device from SuperBASIC.

## 26.33 SDP_KEY

| | |
|---|---|
| Syntax | SDP_KEY [key$] |
| Location | Gold Card, Trump Card, SDUMP_REXT, ST/QL |

In order to facilitate easy screen dumps, the command SDP_KEY will set up a hotkey which when pressed together with <ALT> will cause the whole of the screen starting at $20000 to be sent to the SDUMP device. Under SMS, you will first need to LRESPR SDUMP_REXT provided on the distribution disk to use this command. As with ALTKEY, if the specified key$ is in upper case, you will need to press <ALT><SHIFT> together with the key, or <ALT> with the key if capslock is on. SDP_KEY without any parameters inhibits the hotkey.

**Example**

```
SDP_KEY p
```

will cause the screen to be dumped each time that <ALT><P> is pressed.

**CROSS-REFERENCE**

*SDP_DEV* allows you to alter where the output is to go. See *SDP_SET* and *SDUMP*.

## 26.34 SDP_SET

| | |
|---|---|
| Syntax | SDP_SET printer [,scale [,inverse [,random]]] |
| Location | Gold Card, Trump Card, SDUMP_REXT, ST/QL |

SDP_SET allows you to choose the type of printer attached to the output device, together with how the output is to appear. Under SMS, you will first need to LRESPR SDUMP_REXT provided on the distribution disk to use this command. There are currently 23 types of printer supported, numbered 1…23.

You can also specify the print scale to be used and whether or not the screen is to be printed in inverse colours (by setting the inverse parameter to 1). You can even specify that a random element is to be taken into account in converting the colours to gray shades on the printer (again by setting the random parameter to 1).

The effects of these different parameters all depend upon the printer attached to the output port and the size and shape of the area being dumped. The scale will affect the density of the dots on the printed page. Unfortunately, this does mean that at some of the lower densities, not all of the screen can be printed on an 80 column printer (See the columns headed Max Width in the table below).

If any one of the parameters is not specified, that particular setting will remain unchanged. If you do not have one of the printers currently supported, try out the various dump routines to see which one best suits your needs. For example, users of the Epson Inkjet range of printers will find that the Epson LQ2500 24 pin colour driver is very effective. The range of printers and scales currently supported are detailed in the following tables.

**Note:** In the original manual, this was a single table covering both Mode 4 and Mode 8 screens. Due to the width of a PDF page, the table is far too wide and I've split it into two tables, one for Mode 4 and the other for Mode 8.

| **Mode 4 Screens** | | | | | | |
|---|---|---|---|---|---|---|
| **Printer** | Scale | Dots Per In | Lines Per In | Dot Ratio | Max Width | Ratio |
| 1 Epson MX 80 or similar | 1 | 120 | 72 | 1x1 | 512 | 1.23 |
| | 2 | 60 | 72 | 1x2 | 480 | 1.23 |
| | 3 | 120 | 72 | 2x2 | 480 | 1.23 |

continues on next page

Table 1 – continued from previous page

| **Mode 4 Screens** | | | | | | |
|---|---|---|---|---|---|---|
| **Printer** | Scale | Dots Per In | Lines Per In | Dot Ratio | Max Width | Ratio |
| 2 Epson FX80 additional formats | 1 | 90 | 72 | 1x1 | 512 | 0.92 |
| | 2 | 90 | 72 | 1x1 | 512 | 0.92 |
| | 3 | 90 | 72 | 2x2 | 360 | 0.92 |
| 3 Epson FX100 wide carriage | 1 | 90 | 72 | 1x1 | 512 | 0.92 |
| | 2 | 90 | 72 | 1x1 | 512 | 0.92 |
| | 3 | 90 | 72 | 2x2 | 512 | 0.92 |
| 4 Epson JX80 | 1 | 90 | 72 | 1x1 | 512 | 0.92 |
| | 2 | 90 | 72 | 1x1 | 512 | 0.92 |
| | 3 | 90 | 72 | 2x2 | 512 | 0.92 |
| 5 Epson LQ2500 8 pin | 1 | 80 | 60 | 1x1 | 512 | 0.99 |
| | 2 | 120 | 60 | 2x1 | 512 | 0.74 |
| | 3 | 80 | 60 | 2x2 | 512 | 0.99 |
| 6 Epson LQ2500 24 pin | 1 | 120 | 180 | 1x2 | 512 | 0.99 |
| | 2 | 180 | 180 | 2x3 | 512 | 1.11 |
| | 3 | 180 | 180 | 3x4 | 512 | 0.99 |
| 7 Epson LQ2500 8 pin colour | 1 | 80 | 60 | 1x1 | 512 | 0.99 |
| | 2 | 120 | 60 | 2x1 | 512 | 0.74 |
| | 3 | 80 | 60 | 2x2 | 512 | 0.99 |
| 8 Epson LQ2500 24 pin colour | 1 | 120 | 180 | 1x2 | 512 | 0.99 |
| | 2 | 180 | 180 | 2x3 | 512 | 1.11 |
| | 3 | 180 | 180 | 3x4 | 512 | 0.99 |
| 9 Brother HR4 | 2 | 60 | 72 | 1x2 | 480 | 1.23 |
| | 3 | 120 | 72 | 2x2 | 480 | 1.23 |

continues on next page

Table 1 – continued from previous page

| Mode 4 Screens<br>Printer | Scale | Dots Per In | Lines Per In | Dot Ratio | Max Width | Ratio |
|---|---|---|---|---|---|---|
| 10 Olivetti JP101 | 1 | 110 | 72 | 1x1 | 512 | 1.13 |
| | 2 | 110 | 108 | 1x1 | 512 | 0.75 |
| | 3 | 110 | 72 | 2x2 | 440 | 1.13 |
| 11 Seikosha GP-100A | 1 | 60 | 63 | 1x1 | 480 | 0.70 |
| | 2 | 60 | 63 | 1x2 | 480 | 1.41 |
| 12 Seikosha GP-250X | 1 | 60 | 72 | 1x1 | 480 | 0.61 |
| | 2 | 60 | 72 | 1x2 | 480 | 1.23 |
| 13 Seikosha GP-700A | 1 | 80 | 80 | 1x1 | 512 | 0.74 |
| | 2 | 80 | 80 | 1x2 | 512 | 1.48 |
| | 3 | 80 | 80 | 1x2 | 512 | 1.48 |
| 14 Canon PJ 1080A | 1 | 80 | 80 | 1x1 | 512 | 0.74 |
| | 2 | 80 | 80 | 1x2 | 512 | 1.48 |
| | 3 | 80 | 80 | 1x2 | 512 | 1.48 |
| 15 Centronics 739 | 1 | 75 | 72 | 1x1 | 512 | 0.77 |
| | 2 | 75 | 72 | 1x1 | 512 | 0.77 |
| | 3 | 75 | 72 | 2x2 | 300 | 0.77 |
| 16 C.Itoh 7500 | 1 | 120 | 72 | 1x1 | 512 | 1.23 |
| | 2 | 160 | 72 | 2x1 | 512 | 0.82 |
| | 3 | 120 | 72 | 2x2 | 480 | 1.23 |
| 17 Toshiba TH2100H 24 pin | 1 | 180 | 180 | 1x2 | 512 | 1.48 |
| | 2 | 180 | 180 | 2x3 | 512 | 1.11 |
| | 3 | 180 | 180 | 3x4 | 512 | 0.99 |

continues on next page

Table 1 – continued from previous page

| **Mode 4 Screens** | | | | | | |
|---|---|---|---|---|---|---|
| **Printer** | Scale | Dots Per In | Lines Per In | Dot Ratio | Max Width | Ratio |
| 18 Brother 8056 | 1 | 70 | 72 | 1x1 | 512 | 0.72 |
| | 2 | 70 | 72 | 1x1 | 512 | 0.72 |
| | 3 | 70 | 72 | 2x2 | 280 | 0.72 |
| 19 Epson MX100 or similar | 1 | 120 | 72 | 1x1 | 512 | 1.23 |
| | 2 | 60 | 72 | 1x2 | 512 | 1.23 |
| | 3 | 120 | 72 | 2x2 | 512 | 1.23 |
| 20 Tandy DMP 105 | 1 | 100 | 72 | 1x1 | 512 | 1.03 |
| | 2 | 60 | 72 | 1x2 | 512 | 1.23 |
| | 3 | 100 | 72 | 2x2 | 400 | 1.03 |
| 21 OKI Microline 82/84 OK writer | 1 | 100 | 66 | 1x1 | 512 | 1.12 |
| | 2 | 100 | 66 | 1x1 | 512 | 1.12 |
| | 3 | 100 | 66 | 2x2 | 400 | 1.12 |
| 22 Fastext 80 | 1 | 72 | 72 | 1x1 | 512 | 0.74 |
| | 2 | 60 | 72 | 1x2 | 480 | 1.23 |
| | 3 | 72 | 72 | 2x3 | 288 | 1.11 |
| 23 MT-80 | 1 | 85 | 82 | 1x1 | 512 | 0.77 |
| | 2 | 170 | 82 | 2x1 | 512 | 0.77 |
| | 3 | 170 | 82 | 3x3 | 425 | 1.02 |

| Mode 8 Screens | | | | | | |
|---|---|---|---|---|---|---|
| **Printer** | Scale | Dots Per In | Lines Per In | Dot Ratio | Max Width | Ratio |
| 1 Epson MX 80 or similar | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 60 | 72 | 2x2 | 240 | 1.23 |
| | 3 | 120 | 72 | 4x2 | 240 | 1.23 |
| 2 Epson FX80 additional formats | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 90 | 72 | 2x1 | 256 | 0.92 |
| | 3 | 90 | 72 | 4x2 | 180 | 0.92 |
| 3 Epson FX100 wide carriage | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 90 | 72 | 2x1 | 256 | 0.92 |
| | 3 | 90 | 72 | 4x2 | 256 | 0.92 |
| 4 Epson JX80 | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 90 | 72 | 2x1 | 256 | 0.92 |
| | 3 | 90 | 72 | 4x2 | 256 | 0.92 |
| 5 Epson LQ2500 | 1 | 60 | 60 | 1x1 | 256 | 1.48 |
| 5 Epson LQ2500 8 pin | 2 | 80 | 60 | 2x1 | 256 | 0.99 |
| | 3 | 80 | 60 | 4x2 | 256 | 0.99 |
| 6 Epson LQ2500 24 pin | 1 | 120 | 180 | 1x1 | 256 | 0.99 |
| | 2 | 180 | 180 | 3x3 | 256 | 0.99 |
| | 3 | 180 | 180 | 6x4 | 256 | 0.99 |
| 7 Epson LQ2500 8 pin colour | 1 | 60 | 60 | 1x1 | 256 | 1.48 |
| | 2 | 80 | 60 | 2x1 | 256 | 0.99 |
| | 3 | 80 | 60 | 4x2 | 256 | 0.99 |

continues on next page

Table 2 – continued from previous page

| **Mode 8 Screens** | | | | | | |
|---|---|---|---|---|---|---|
| **Printer** | Scale | Dots Per In | Lines Per In | Dot Ratio | Max Width | Ratio |
| 8 Epson LQ2500 24 pin colour | 1 | 120 | 180 | 1x1 | 256 | 0.99 |
| | 2 | 180 | 180 | 3x3 | 256 | 0.99 |
| | 3 | 180 | 180 | 6x4 | 256 | 0.99 |
| 9 Brother HR4 | 1 | 120 | 72 | 1x1 | 512 | 1.23 |
| | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 60 | 72 | 2x2 | 240 | 1.23 |
| | 3 | 120 | 72 | 4x2 | 240 | 1.23 |
| 10 Olivetti JP101 | 1 | 110 | 108 | 1x1 | 256 | 0.75 |
| | 2 | 110 | 108 | 3x3 | 256 | 1.00 |
| | 3 | 110 | 72 | 4x2 | 220 | 1.13 |
| 11 Seikosha GP-100A | 1 | 60 | 63 | 1x1 | 256 | 1.41 |
| | 2 | 60 | 63 | 2x2 | 240 | 1.41 |
| 12 Seikosha GP-250X | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 60 | 72 | 2x2 | 240 | 1.23 |
| 13 Seikosha GP-700A | 1 | 80 | 80 | 1x1 | 256 | 1.48 |
| | 2 | 80 | 80 | 2x2 | 256 | 1.48 |
| | 3 | 80 | 80 | 3x3 | 212 | 0.99 |
| 14 Canon PJ 1080A | 1 | 80 | 80 | 1x1 | 256 | 1.48 |
| | 2 | 80 | 80 | 2x2 | 256 | 1.48 |
| | 3 | 80 | 80 | 3x3 | 212 | 0.99 |
| 15 Centronics 739 | 1 | 75 | 72 | 1x1 | 256 | 1.42 |
| | 2 | 75 | 72 | 2x1 | 256 | 0.77 |
| | 3 | 75 | 72 | 3x3 | 200 | 1.03 |

continues on next page

Table 2 – continued from previous page

| **Mode 8 Screens** | | | | | | |
|---|---|---|---|---|---|---|
| **Printer** | Scale | Dots Per In | Lines Per In | Dot Ratio | Max Width | Ratio |
| 16 C.Itoh 7500 | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 120 | 72 | 2x1 | 256 | 1.23 |
| | 3 | 120 | 72 | 4x2 | 240 | 1.23 |
| 17 Toshiba TH2100H 24 pin | 1 | 180 | 180 | 2x2 | 256 | 1.48 |
| | 2 | 180 | 180 | 3x3 | 256 | 0.72 |
| | 3 | 180 | 180 | 6x4 | 256 | 0.99 |
| 18 Brother 8056 | 1 | 70 | 72 | 1x1 | 256 | 1.44 |
| | 2 | 70 | 72 | 2x1 | 256 | 0.72 |
| | 3 | 70 | 72 | 3x3 | 186 | 0.96 |
| 19 Epson MX100 or similar | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 60 | 72 | 2x2 | 256 | 1.23 |
| | 3 | 120 | 72 | 4x2 | 256 | 1.23 |
| 20 Tandy DMP 105 | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 100 | 72 | 2x1 | 256 | 1.03 |
| | 3 | 100 | 72 | 4x2 | 200 | 1.03 |
| 21 OKI Microline 82/84 OK Writer | 1 | 60 | 66 | 1x1 | 256 | 1.35 |
| | 2 | 100 | 66 | 2x1 | 256 | 1.12 |
| | 3 | 100 | 66 | 4x2 | 200 | 1.12 |
| 22 Fastext 80 | 1 | 60 | 72 | 1x1 | 256 | 1.23 |
| | 2 | 60 | 72 | 2x2 | 240 | 1.23 |
| | 3 | 72 | 72 | 3x3 | 192 | 0.99 |
| 23 MT-80 | 1 | 85 | 82 | 1x1 | 256 | 1.53 |
| | 2 | 170 | 82 | 3x1 | 256 | 1.02 |
| | 3 | 170 | 82 | 6x2 | 212 | 1.02 |

The resultant dump will depend both on the current screen mode and the chosen scale. The dot ratio column shown above represents the size of the resultant picture as a ratio of the original. For example, if the Dot ratio is 1x1 and you are outputting a screen of 512x256 pixels at 120 dots per inch and 72 lines per inch, you can expect the resultant picture to be 512/120 inches across by 256/72 inches down. If however, the Dot ratio was 1x2 (with the same number of dots per inch and lines per inch as above), then the resultant picture will be 512/120 inches across by 2*256/72 inches down.

The ratio column in the above table shows the resultant ratio between the vertical size/horizontal size. The nearer that this ratio is to 1.00, the more circular your screen circles will appear on paper. The default is printer 1, scale 1, inverse 1, random 0.

**NOTE**

There is no check on the parameters, other than to ensure that there are the correct number of parameters.

**CROSS-REFERENCE**

*SDUMP* actually prints the screen using the chosen format.

## 26.35 SDUMP

| | |
|---|---|
| Syntax | SDUMP #ch or<br>SDUMP [width,height,xpos,ypos] or<br>SDUMP [{address \| address,width,height,xpos,ypos}] |
| Location | Gold Card, Trump Card, SDUMP_REXT, ST/QL |

The command SDUMP allows you to dump a screen (or part of a screen) to a printer (or a file), using one of the in-built formats (one of which will hopefully work on your printer!) - see SDP_SET.

Under SMS, you will first need to LRESPR SDUMP_REXT provided on the distribution disk to use this command.

The first variant is the simplest, it will dump the whole of the contents of the specified window #ch to the printer. If the second variant is used, SDUMP will dump the whole of the screen defined by widthXheightAxposXypos (using absolute pixel co-ordinates).

If no parameters are supplied, SDUMP will dump the whole screen. The third variant of the command is intended to dump a screen which has been stored under the Pointer Environment's PSAVE function. The address returned by PSAVE should be used as the first parameter of the SDUMP command. If no further parameters are specified, the whole area stored at the specified address will be dumped, otherwise you can specify the area of that buffer to be dumped in much the same way that you can specify an area of the screen to be dumped.

**Example**

```
OPEN #3,scr_448x200a32x16:SDUMP #3:CLOSE #3
```

and:

```
SDUMP 448,200,32,16
```

are the same.

**NOTE 1**

Some early versions of SDUMP expect the screen to start at 131072 and be 512x256 pixels in size and can therefore get very confused in dual screen mode on Minerva and Amiga QDOS. However, later versions supplied with SMS check the screen size and base when the toolkit is linked into memory and expect it to remain the same afterwards!

**NOTE 2**

SDUMP does not work on Minerva, unless you have v2.23 (or later) of the Trump Card / Gold Card. If you have an earlier version of Toolkit II and want to use SDUMP, you have to ensure that another Job (such as FSERVE) is running when SDUMP is issued.

**NOTE 3**

Once SDUMP has started its work, it is not easy to abort it early - any further attempt to use the serial port will result in the error 'In Use'.

**NOTE 4**

If you have directed the output to a file, the file will be automatically overwritten if necessary.

**CROSS-REFERENCE**

*SDP_SET* allows you to alter the printer format. *SDP_KEY* allows you to set up a hotkey to dump the screen. *SDP_DEV* allows you to alter the device where the dump is to be sent.

## 26.36 SEARCH

| | |
|---|---|
| Syntax | SEARCH (add1 TO add2, tofind$) or<br>SEARCH (add1 TO add2, tofind$ [!] )(BTool only) |
| Location | TinyToolkit, BTool |

This function scans RAM memory from address add1 to add2 for the given string tofind$ and returns the address of its first occurrence or zero if it was not found. The search is not case-dependent in the Tiny-Toolkit version whilst BTool introduces an optional switch: a '!' after tofind$ disables case-sensitivity and reduces speed.

**Example**

The following small program will scan the whole memory, ROM included, from adr onwards for string$. Tiny-Toolkit SEARCH is assumed, PHYSTOP is also necessary:

```
100 string$="dev v" : REMark what we are looking for
110 add=0 : REMark start address
120 MODE 4: CSIZE 0,0: PAPER 0: INK 5: CLS
130 REPeat searching
140   add=SEARCH(add+1,PHYSTOP-add,string$)
150   IF NOT add THEN EXIT searching
160   PRINT \"Address ="!add
170   PRINT PEEK$(add-20,19);
180   INK 7: PRINT PEEK$(add,LEN(string$));
190   INK 5: PRINT PEEK$(add+LEN(string$),20)
200 END REPeat searching
210 PRINT "That's all."
```

**NOTE**

The search string tofind$ will always be found at least twice in memory because tofind$ itself needs to be stored somewhere.

**CROSS-REFERENCE**

*PEEK$*, *DEV_USE*. See other implementation of *SEARCH*. See *MSEARCH*, *SEARCH_MEM* and *TTFINDM* also.

## 26.37 SEARCH

| | |
|---|---|
| Syntax | SEARCH (array$, tofind$, start, compare [,row] ) |
| Location | ARRAY |

The function SEARCH searches in a two or three-dimensional string array array$ for the string tofind$. The search is not case-sensitive but nevertheless very fast (as the example shows). SEARCH will always look at one row only - there is just one if the array is two-dimensional but for three-dimensional string arrays (where there are in fact two-dimensions of strings because the third dimension is the maximum string length) the optional row parameter which defaults to the first row can be used to select a certain row. The start parameter allows you to tell SEARCH from which element in the row onwards it should look (remember that the first element is indexed with 0). Compare specifies the number of characters at the start of each entry to ignore, so 0 will search the whole entry for tofind$. The search stops if tofind$ was found in an entry but not if the entry and tofind$ are identical.

SEARCH returns the entry index or -1 if no suitable entry was found.

**Example**

Lines 100 to 170 of the following example initialise the name$ array with n (here 1000) random strings of varying length, from four to 10 characters; this can take a while. After that, the whole array is scanned for the string QL and all occurrences are listed. If you want to check out the tremendous speed of SEARCH, amend line 100, set n to 10000 and assure that at least 100K of memory is free for the huge array: you will be surprised, even the 10000 entries are searched in next to no time!

```
100 n = 1000: DIM name$(n,10)
110 FOR i = 1 TO n
120   name$(i) = ""
130   FOR j = 1 TO 10
140     name$(i) = name$(i) & CHR$(RND(65 TO 90))
150     IF j > 3 AND NOT RND(5) THEN EXIT j
160   END FOR j
170 END FOR i
180 :
190 first = 1
200 REPeat loop
210   found = SEARCH(name$, "QL", first, 0)
220   IF found < 0 THEN EXIT loop
230   PRINT name$(found)
240   IF found = n THEN EXIT loop: ELSE first = found + 1
250 END REPeat loop
```

Minerva and SMS users can use integers for n, i, j, first and found to speed up things, so replace them by n%, i%, j%, first% and found%.

**CROSS-REFERENCE**

Use *INSTR* to locate a sub-string in a string. *INARRAY%* is similar. See the other implementation of *SEARCH*.

## 26.38 SEARCH_C

| | |
|---|---|
| Syntax | address = SEARCH_C(start, length, what_for$) |
| Location | DJToolkit 1.16 |

See *SEARCH_I* for details.

**CROSS-REFERENCE**

*SEARCH_I*.

## 26.39 SEARCH_I

| | |
|---|---|
| Syntax | address = SEARCH_I(start, length, what_for$) |
| Location | DJToolkit 1.16 |

This function, and *SEARCH_C* above, search through memory looking for the given string. *SEARCH_C* searches for an EXACT match whereas SEARCH_I ignores the difference between lower & UPPER case letters.

If the address returned is zero, the string was not found, otherwise it is the address where the first character of what_for$ was found, or negative for any errors that may have occurred.

If the string being searched for is empty ("") then zero will be returned, if the length of the buffer is negative or 0, you will get a 'bad parameter' error (-15). The address is considered to be unsigned, so negative addresses will be considered to be very large positive addresses, this allows for any future enhancements which will allow the QL to use a lot more memory than it does now!

**EXAMPLE**

```
1000 PRINT SEARCH_C(0, 48 * 1024, 'sinclair')
1010 PRINT SEARCH_I(0, 48 * 1024, 'sinclair')
1020 PRINT
1030 PRINT SEARCH_C(0, 48 * 1024, 'Sinclair')
1040 PRINT SEARCH_I(0, 48 * 1024, 'Sinclair')
```

The above fragment, on my Gold Card JS QL, prints:

```
0
47314

47314
47314
```

Looking into the ROM at that address using

```
PEEK_STRING(47314, 21)
```

gives:

```
Sinclair Research Ltd
```

which is part of the copyright notice that comes up when you switch on your QL. The reason for zero in line 1000 is because the 's' is lower case, case is significant and the ROM has a capital 'S', so the text was not found in the ROM.

**CROSS-REFERENCE**

*SEARCH_C*.

## 26.40 SEARCH_MEM

| | |
|---|---|
| Syntax | SEARCH_MEM (add1 TO add2, tofind$) |
| Location | MSEARCH (DIY Toolkit - Vol X) |

This function is very similar to the main MSEARCH function provided by this toolkit. It is however limited to case-dependent searches and therefore is even quicker than MSEARCH.

**CROSS-REFERENCE**

See *SEARCH* and *TTFINDM* also. *MSEARCH* is a variant on this version.

## 26.41 SELect

| | |
|---|---|
| Syntax | SELect |
| Location | QL ROM |

This keyword forms an integral part of the SELect ON structure identifier and has no use on its own. If you try to enter it on its own, the error 'Bad Name' will be generated.

**CROSS-REFERENCE**

Please see *SELect ON*!

## 26.42 SELect ON

| | |
|---|---|
| Syntax | SELect ON var |
| Location | QL ROM |

This command is used to mark the start of a SuperBASIC structure which is an extremely quick means of testing for various values of a variable and taking a different course of action in a program according to those values. Unfortunately, the standard form of this command only allows you to test for different values of a numeric variable (eg. SELect ON a$ is not allowed).

There are actually two forms of the SuperBASIC structure:

SELect ON var=range: statement *[:statement]**[:=range:statement *[:statement]*]*

or

SELect ON var *[[ON var] = range:statement *[:statement]* ]* .. END SELect

Range can be any one, or mixtures of, the following:

- Expression
- Expression TO Expression
- REMAINDER

The first of these two SELect variants (in this and all SuperBASIC structures) is known as an in-line structure, as the entire structure appears on the same program line. This does not need END SELect to mark the end of the structure.

After the main SELect ON var statement, the interpreter looks for a list of possible values, and then if the value of the given variable falls within the range of possible values, the program takes action according to the statements which follow that value in the list.

The interpreter will use the first range of values into which it can fit the variable and once found, all statements up until (but excluding) the next range in the list will be treated as applying to that range (whether they appear on the same line or not). Once all of the statements applying to that range have been executed, control passes to the statement following the END SELect statement (or if the in-line form of the structure is used, and END SELect does not appear on that line, then control passes to the next line).

The way in which matches are made when checking whether a value falls within a range depends on whether range is a single number eg:

```
ON var = 100
```

or various values eg:

```
ON var = 90 TO 100
```

If the former, the value need only be approximately equal to range (ie. to within 1 part in $10^7$, for instance: 100.0000045==100!). However, if the latter format is used, a match will only be found if the given value is within the absolute range (eg. in the above example, 100.0000045 would not be matched!).

If the long form of the structure is used, and ON var is used within the body of the structure, this must be the same variable as that used in the initial SELect ON statement.

**Example 1**

```
10 SELect ON x=1,10 TO 100,500:PRINT 'x'
```

**Example 2**

```
100 SELect ON test
110   = 0,2,4,6,8,10: PRINT 'Even Number'
120   = REMAINDER: PRINT 'Odd Number'
130 END SELect
```

**Example 3**

A re-write of the example given for ON…GO SUB:

```
100 no_of_locations=3
110 start=0
120 PRINT_LOC 2
125 :
130 DEFine PROCedure PRINT_LOC(xa)
135   xa=xa+start
140   SELect ON xa
150     = 1: PRINT 'This is location 1'
160     =2
165       PRINT 'This is location 2'
170     =3: PRINT 'This is location 3'
180     = REMAINDER: PRINT 'Undefined Location'
185       RETurn
190   END SELect
200   PRINT 'What now?:RETurn
210 END DEFine
```

**NOTE 1**

Pre JS ROMs and SMS allow you to enter string and integer variables into the SELect statement, but they will not work unless you used a SuperBASIC compiler. Later ROMs, report a 'bad line' error unless you have Minerva.

**NOTE 2**

On JS ROMs, you cannot use a parameter passed to a PROCedure or FuNction as the variable in a SELect ON statement unless it appears as the last parameter in the list in the definition line. If you do try to break this rule, you will end up with a 'bad name' error. The answer is to copy the parameter to a temporary variable.

**NOTE 3**

As you may have noticed, unlike other SuperBASIC structures which will expand a command typed into the full structure name if you type just the capital letters (eg. DEFPROC becomes DEFine PROCedure), SELON will not be expanded to SELect ON. You will need to type SEL ON instead.

**NOTE 4**

To maximise the speed of the SELect ON command, ensure that the most common matches appear at the start of the definition block.

**NOTE 5**

Except under SMS, SELect ON can only cope with simple variables, for example:

```
SELect ON a
```

is acceptable. Compare:

```
SELect ON a(2)
SELect ON s*10
SELect ON CODE(a$)
```

All of these are acceptable on SMS but cannot currently be compiled.

Although lines such as:

```
SELect ON CODE
```

and:

```
SELect ON INKEY$
```

might be accepted by the interpreter, the lines contained within the block will be ignored (other than =REMAINDER matches). On SMS both of these give an 'error in expression' when RUN.

**MINERVA NOTES**

Minerva supports string and variables in SELect ON statements. The check for characters is normally case independent. For example:

```
SELect ON a$: ='hello'
```

will find both a$='HeLLo' and a$='hello'. If however, you want the match to be exact (case dependent), then something along the lines of:

```
SELect ON a$: ='hello' TO 'hello'
```

must be used. Unfortunately, you still cannot SELect ON machine code functions (for example, INKEY$), which will have no effect, or slice the string, which will cause a 'bad line' error. A short example of the additional flexibility is a check for a response to a simple question {eg. Overwrite (y/n)?}:

```
100 REPeat loop
110   A$=INKEY$(-1)
120   SELect ON A$
130     ='yn'&chr$(27):EXIT loop
140   END SELect
150 END REPeat loop
```

is the same as:

```
100 REPeat loop
110   A=CODE(INKEY$(-1))
120   SELect ON A:
130     =89,121,78,110,27:EXIT loop
140   END SELect
150 END REPeat loop
```

Minerva also supports integer variables, such as:

```
SELect ON a%
```

This is an extremely fast means of testing a condition. However, due to the nature of integers, tests will only match the integer part of range.

**SMS NOTE**

This has greatly extended the flexibility of SELect ON - see in particular Note 5 above. It will also allow integer variables as the SELect, but unfortunately not string SELect variables at present. If you try to do

so, the error 'Incorrectly structured SELect clause' will be reported. It will however, even support things like:

```
SELect ON CODE(INKEY$(#1))
```

Unfortunately, SMS pre v2.90 had problems in dealing with in-line SELect ON statements. Prior to v2.89 an error would be generated if an END SELect statement did not appear in an in-line definition, and v2.89 reported an error if END SELect did appear!!

**CROSS-REFERENCE**

A slower means of testing for values is the structure *IF* ... *END IF*. *END SELect* ends a *SELect ON* structure.

## 26.43 SEND_EVENT

| | |
|---|---|
| Syntax | SEND_EVENT {jobname$ \| jobID \| jobnr,tag }, event |
| Location | SMSQ/E v2.71+ |

With v1.51 of the Window Manager (and v2.71 of SMSQ/E), the possibility of Job Events was introduced. This is basically a simple way of making one program wait until it receives notification from another Job that up to eight different events has occurred.

The events are undefined and simply represented by the eight numbers : 1, 2, 4, 8, 16, 32, 64, 128. This command allows you to tell a specified job that those events have occurred - several events may be notified by adding together the various values of event. The job to be notified can be represented by either its:

1. Jobname (eg. 'SBASIC')
2. Job ID number (returned by OJOB for example).
3. Job number and Job Tag (returned by JOBS).

**Example**

```
SEND_EVENT OJOB(1), 2+8
```

Notifies the current job's owner that events 2 and 8 have occured.

**CROSS-REFERENCE**

A job can test to see if an event has occured with *WAIT_EVENT*.

## 26.44 SERMAWS

| | |
|---|---|
| Syntax | SERMAWS acc%, wup% |
| Location | SERMouse |

This command is used to set two parameters which control the effect that moving the serial mouse has on the on-screen pointer under the Pointer Environment. The first parameter sets the speed at which the pointer will accelerate across the screen (this can be any value in the range 0...9). A standard value is 6. The second parameter sets the initial speed of the pointer. A standard value is 3. The values can also be set by configuring the SERMouse file.

**CROSS-REFERENCE**

*SERMPTR* makes the mouse driver affect the Pointer only. Qpac 2 allows you to set the same parameters from the Sysdef menu. Also refer to*SERMSPEED* and *SERMON*. See the appendix on Mouse Drivers for more information.

## 26.45 SERMCUR

| | |
|---|---|
| Syntax | SERMCUR |
| Location | SERMouse |

The SERMouse driver allows you to use a Mouse to control either the Pointer (under the Pointer Environment) or the Basic cursor (used in INPUT commands or similar). This command forces the mouse to control the Basic cursor provided that the following condition is met: There is a channel currently open which is awaiting for screen input with a visible cursor. If you switch to a program which is reading the pointer (ie. a program which uses the pointer interface) then the command SERMPTR is automatically called.

**CROSS-REFERENCE**

*SERMPTR* switches to Pointer Mode. See also *SERMSPEED*. You can also switch to cursor mode by hitting the left hand mouse button twice in quick succession.

## 26.46 SERMOFF

| | |
|---|---|
| Syntax | SERMOFF |
| Location | SERMouse |

This command removes the Serial Mouse Driver.

**CROSS-REFERENCE**

*SERMON* will reactivate the Driver. Compare *SERMWAIT*

## 26.47 SERMON

| | |
|---|---|
| Syntax | SERMON |
| Location | SERMouse |

The serial mouse driver must always be loaded into Resident Procedure Space (for example with RESPR or LRESPR) before any Jobs are EXECuted. However, if you have Hermes or SuperHermes fitted, you can configure the Serial Mouse driver so that it does not automatically start up after being linked into BASIC.

This command can be used to initialise and startup the driver either following a SERMOFF command or if you have configured the driver not to automatically start up after being linked into BASIC. SERMON should also be used to reactivate the driver following a SERMWAIT command.

**CROSS-REFERENCE**

*SERMOFF* and *SERMWAIT* are complementary functions. See the Appendix on Mouse Drivers for further details.

## 26.48 SERMPTR

| | |
|---|---|
| Syntax | SERMPTR |
| Location | SERMouse |

This command switches the Serial Mouse Driver into Pointer Mode, so that the movements of the Serial Mouse affect the Pointer on screen, allowing you to control programs which make use of the Pointer Environment. This is the default mode following loading the driver or a SERMON command.

**CROSS-REFERENCE**

See also *SERMCUR*.

## 26.49 SERMRESET

| | |
|---|---|
| Syntax | SERMRESET |
| Location | SERMouse |

This command should never really be needed, particularly if you are using the Serial Mouse with Hermes or SuperHermes. This command resets the chip which controls the serial ports and should only be necessary if you notice the Pointer or Cursor moving on screen uncontrollably.

**CROSS-REFERENCE**

Other causes of this problem may be the wrong speed settings - see *SERMAWS*, *SERMSPEED* and *BAUD*.

## 26.50 SERMSPEED

| | |
|---|---|
| Syntax | SERMSPEED mul%, div%, acc% [,cursormul%, cursordiv%] |
| Location | SERMouse |

This command allows you to set various parameters to dictate the speed and resolution of the mouse. As a mouse moves, it sends a stream of data to the computer containing details of the direction moved and the distance moved. These details are sent every few microseconds and converted by the driver to x,y coordinates on screen. The speed at which these details are sent is known as the resolution of the mouse.

This command allows you to alter the resolution of the mouse so that you do not have to move the mouse as far to get the pointer (or cursor) on screen to move across the whole screen.

The mul% and div% parameters can be in the range 0..127 (with 0 disabling this feature - the default).

The distance sent by the mouse is multiplied by the mul% factor and divided by the div% factor - with these both set to 0, only two-thirds of the distance moved by the mouse is passed to the Pointer Interface to be translated into movements of the Pointer.

The acc% parameter can be in the range (0…8) and defaults to 4 - this is used to calculate an acceleration factor, so that the faster that the mouse is moved the quicker the details sent by the mouse are passed on to the Pointer Interface (thus making the Pointer move in bigger and bigger steps).

The last two parameters are optional and are only relevant when the Mouse Driver is used in Cursor Mode (see SERMCUR). These two parameters affect the resolution of the mouse when being used to move the Basic Cursor - the standard values are both 1.

**CROSS-REFERENCE**

*SERMAWS* works in conjunction with this command. All of these parameters can be configured in the SERMouse file. See also *SERMCUR* and *SERMPTR*.

## 26.51 SERMWAIT

| | |
|---|---|
| Syntax | SERMWAIT |
| Location | SERMouse |

This command can be used to suspend the Serial Mouse Driver. You may wish to do this for example, if your system does not support dual BAUD rates and you need to change the baud rate for a Modem or Printer.

**CROSS-REFERENCE**

*SERMON* re-activates the Driver. Again, the Driver can be configured to automatically be suspended when the baudrate is altered.

## 26.52 SERNET

| | |
|---|---|
| Syntax | SERNET |
| Location | SMSQ/E, ATARI Emulators |

A file SERNET_rext is provided with SMSQ/E, QXL and the Emulators for the Atari computers which allows you to set up a Network using the Serial ports provided. Once the Network has been set up with the necessary leads, and SERNET_rext been loaded on all computers in the Network, the command SERNET should be issued to start up the fileserver Job on each computer. This creates a background Job called 'SERNET' which is similar to the 'Server' Job created by FSERVE.

The two fileservers are very similar in operation in that they both allow other computers to access the resources of the Master machine over the Network. As with MIDINET, SERNET has built-in protection for files which can prevent other users in a Network accessing sensitive files. Refer to MIDINET for details.

**CROSS-REFERENCE**

*SNET* is needed to control the Network. See also *FSERVE* and *MIDINET*. See the Appendix on Networks for further details.

## 26.53 SER_ABORT

| | |
|---|---|
| Syntax | SER_ABORT [port] |
| Location | ST/QL, SMSQ/E |

This command is similar to PAR_ABORT except that it clears out all of the closed SER buffers and then sends an 'aborted' message, to the SER device. If port is specified, on machines which support more than one serial port, this allows you to specify the port number to be affected (default SER1).

**CROSS-REFERENCE**

See *PAR_ABORT*.

## 26.54 SER_BUFF

| | |
|---|---|
| Syntax | SER_BUFF [port, ] output_size [,input_size] |
| Location | ST/QL, SMSQ/E |

Used with one parameter, this is the same as PAR_BUFF except that it sets the size of the output buffer attached to each SER channel. The input buffer is normally a dynamic buffer, unless input_size is specified. The output buffer should be a minimum of 5 to avoid confusion with the port number.

You can also use SER_BUFF to alter the size of the input buffer, by using the form:

```
SER_BUFF output_size, input_size
```

Although you will need to specify the output_size, you can set this to 0 to enable a dynamic output buffer. You can also specify which serial port number is to be used to allow this command to work on machines with more than one serial port (this defaults to SER1).

**Examples**

```
SER_BUFF 200
```

Set the output buffer size to 200 bytes, with a dynamic input buffer.

```
SER_BUFF 200,500
```

Have an output buffer of 200 bytes, with an input buffer of 500 bytes.

**NOTE 1**

The actual usable input buffer will be calculated by the value set by SER_BUFF less the value set by SER_ROOM.

**NOTE 2**

In version E-17 of the device drivers for the Atari Emulator (and later implementations of this command, including SMSQ/E), whenever you use this command, the value set by SER_ROOM

is re-calculated so that it is set to one quarter of the input buffer size. Earlier versions may (after Level B09) would report an error if the input buffer was not at least twice the size of the value set by SER_ROOM.

**CROSS-REFERENCE**

See *PAR_BUFF*! You should also refer to *SER_ROOM*.

## 26.55 SER_CDEOF

| | |
|---|---|
| Syntax | SER_CDEOF [port,] time |
| Location | ST/QL (Level D00 +), SMSQ/E |

Serial ports are able to both send and receive data. It is therefore imperative that the System can detect when data is no longer being sent to a port which is being used to receive the data. Normally, the System will wait until it receives an End Of File character (CTRL Z or EOF). However, it can be useful to specify a time limit, whereby if no data is received during that time, the System assumes End Of File.

The command SER_CDEOF time allows you to specify the number of frames for which the System will wait for more data. If time equals 0, then the System will wait indefinitely until it receives an explicit End Of File character.

The time should be more than 5 in order to distinguish it from the port number. For machines with more than one serial port, you can specify the number of the serial port this command is to apply to (default SER1).

**NOTE 1**

This command has no effect on a QL, QPC or QXL.

**NOTE 2**

This command would not work properly on SCC ports on the Atari Mega STE or TT until v2.73+.

**CROSS-REFERENCE**

*EOF* and *EOFW* allow you to detect an EOF character.

## 26.56 SER_CLEAR

| | |
|---|---|
| Syntax | SER_CLEAR [port] |
| Location | ST/QL, SMSQ/E |

This is similar to PAR_CLEAR except that it clears out all current SER buffers. For machines with more than one serial port, you can specify the number of the serial port to be affected (default SER1).

**CROSS-REFERENCE**

See *PAR_CLEAR*!

## 26.57 SER_FLOW

| | |
|---|---|
| Syntax | SER_FLOW [port,] flow |
| Location | ST/QL, SMSQ/E |

Because of the variety of equipment which can be connected to a QL system through a serial port, the System has to support several types of handshaking. Handshaking is basically a means of checking if the data received through a serial port is the same as the data which has been sent. Normally, handshaking can be specified when a port is opened (see the Appendix concerning device drivers). However, it can also be useful to preset the handshaking by using the command SER_FLOW. flow can have one of three values:

- h Enable handshaking
- i Ignore handshaking - do not bother to check data
- x XON/XOFF detection.

To enable flexibility on machines with more than one serial port, you can also specify the number of the serial port to be affected by this command (default SER1).

**CROSS-REFERENCE**

Please refer to the Appendix on device drivers for more information.

## 26.58 SER_GETPORT$

| | |
|---|---|
| Syntax | com$ = SER_GETPORT$(port%) |
| Location | SMSQ/E for QPC |

Returns the device the SER port is connected to, for example "COM1".

**CROSS-REFERENCE**

See *SER_SETPORT*.

## 26.59 SER_PAUSE

| | |
|---|---|
| Syntax | SER_PAUSE [port,] time |
| Location | SMSQ/E for Gold Card |

On standard QL serial ports, you may find that some characters which are sent by the QL through the serial ports get lost or the device to which they are sent (for example a printer) prints undefined characters.

This problem may be caused by the fact that the stop bit which is sent by the QL serial ports may be too short for the device at the other end.

The SER_PAUSE command allows you to set the length of the stop bit in microseconds - it effectively causes a short pause between each character sent through the serial ports. If port is not specified, this command will affect both serial ports, otherwise it will only affect the specified serial port. The higher the value of time, the longer the stop bit will be and hence the slower the serial transfer rate.

**CROSS-REFERENCE**

If you are using serial ports to receive data, you may need to set *SER_ROOM*. *BAUD* also affects the serial transfer rate. Please also refer to the Appendix on device drivers for more information.

## 26.60 SER_ROOM

| | |
|---|---|
| Syntax | SER_ROOM [port,] bytes |
| Location | ST/QL, SMSQ/E |

Although handshaking should ensure that serial input is safe, unfortunately some devices carry on sending data even though they have been told to stop. This may be caused by a buffer attached between the sending and receiving equipment, for example. This is known as 'serial overrun' and can have unfortunate consequences, as the receiving equipment may not have room to store the additional information.

Where the system is acting as the receiver, you can use the command SER_ROOM to specify a minimum amount of memory which must be left in the input buffer when the System uses handshaking to check on the validity of the data received. SER_ROOM sets aside bytes in the input buffer which can be used to store information received after the System has told the sending equipment to stop.

If you still find that some data is lost due to serial overrun, try increasing the amount of space. For machines with more than one serial port, you can specify the number of the serial port to be affected by this command (default SER1).

**NOTE**

The default room is 32 bytes.

**CROSS-REFERENCE**

*SER_BUFF* allows you to alter the size of the input buffer and affects the value set by this command. You should also look at *SER_PAUSE*.

## 26.61 SER_SETPORT

| | |
|---|---|
| Syntax | SER_SETPORT port%, com$ |
| Location | SMSQ/E for QPC |

Sets the COM port a SER port should be connected with. The change will take effect on the next open of the specified serial port.

**Example**

```
SER_SETPORT 4,"COM32"
```

Will associate SER4 with COM32.

**CROSS-REFERENCE**

See *SER_GETPORT$*.

## 26.62 SER_USE

| | |
|---|---|
| Syntax | SER_USE [device] |
| Location | ST/QL, SMSQ/E |

As with PAR_USE, this command allows the SER port to emulate the parallel printer port. Any three letter extension is allowed, you are not restricted to SER or PAR.

**CROSS-REFERENCE**

See *PAR_USE*

## 26.63 SET

| | |
|---|---|
| Syntax | SET x, y, col |
| Location | HCO |

SET does the same as PLOT with SCRBASE 131072 set, ie. it does not support virtual screens. x ranges from 0 to 511, y from 0 to 255. The colour (col) is specified by an integer from 0 to 3, representing the four colours available in MODE 4: 0 … black 1 … red 2 … green 3

**NOTE**

Although SET is not designed to, it does work in MODE 8 but the colours appear differently: Colour 1 is not red but magenta for example.

**WARNING 1**

SET writes directly into screen memory and assumes that it starts at 131072, so SET may crash the machine if the screen is located at another position in memory. SET also assumes a resolution of 512 x 256 pixels.

**WARNING 2**

SET does not check for the existence of the parameters (this means for example that it will not report 'bad parameter' for SET x, y), it may crash if any of the parameters are omitted.

**CROSS-REFERENCE**

*PLOT*. We highly recommend that you use the QDOS inbuilt window relative graphic routine, *POINT* in this case. *COL* finds the colour of a screen pixel. See the other implementation of *SET* also.

## 26.64 SET

| | |
|---|---|
| Syntax | SET [#]variable TO value |
| Location | SET, ALTER (DIY Toolkit - Vol U) |

This command allows you to set up various universal constants which allow programs to read various values which are set by other programs. This is similar to creating machine code functions which return constant values.

The constants to be set up appear as 'variable' in the command syntax above.

They can be string, floating point or integer but must not have previously been used in the program (otherwise the error 'In Use' will be reported). They must also not appear in quotes. The constants should be SET from SuperBASIC Job 0, otherwise they do not seem to work (at least on Minerva). However, other programs can use ALTER to change the value of the constants and also read the constants as if they were predefined variables.

As an added bonus, if the variable is prefixed by a hash sign, then this is taken to be a pointer to a system variable, which will always point to that system variable even if the system variables move. For example to read the Network number, you could use:

```
SET #NET_ID TO HEX('37')
PRINT PEEK (NET_ID)
```

instead of:

```
SET NET_ID TO HEX('37')
PRINT PEEK (SYS_VARS + NET_ID)
```

**Example**

Set the following from SuperBASIC:

```
10 SET FALSE TO 0 : SET TRUE TO 1
20 SET YES$ TO 'Yy' : SET NO$ TO 'Nn'
30 SET DEF_DRIVE$ TO 'flp1_'
```

Any other program can then just use lines such as:

```
IF INKEY$(1) INSTR YES$ : PRINT 'Yes has been selected'
```

and:

```
LBYTES DEF_DRIVE$ & 'prog_data', space
```

**NOTE 1**

SET does not work on SMS.

**NOTE 2**

SET #value does not appear to work on Minerva v1.97 (at least in versions up to v1.66 of the toolkit).

**NOTE 3**

Any attempt to use SET from within a multiple BASIC will have no effect.

**CROSS-REFERENCE**

See *ALTER*. *TRUE%*, *FALSE%* and *PI* are predefined constants.

## 26.65 SetHEAD

| | |
|---|---|
| Syntax | SetHEAD #ch, adr |
| Location | HEADER (DIY Toolkit - Vol F) |

The command SetHEAD is the counterpart of GetHEAD and is normally used in conjunction with it. So please refer to GetHEAD for further information about the syntax and usage. There is just one difference you must keep in mind: whilst GetHEAD does not care in which mode (read only or read and write) a file was opened, SetHEAD does. It expects that the channel was opened with OPEN, FOPEN etc but not with OPEN_IN or FOP_IN.

**CROSS-REFERENCE**

See *GetHEAD*.

## 26.66 SET_HEADER

| | |
|---|---|
| Syntax | error = SET_HEADER(#channel, buffer) |
| Location | DJToolkit 1.16 |

This function returns the error code that occurred when trying to set the header of the file on the given channel, to the contents of the 64 byte buffer stored at the given address. If the result is zero then you can assume that it worked ok, otherwise the result will be a negative QDOS error code. On normal QLs, the three dates at the end of a file header cannot be set.

**EXAMPLE**

See the example for *READ_HEADER*.

**CROSS-REFERENCE**

*READ_HEADER*.

## 26.67 SET_CLOCK

| | |
|---|---|
| Syntax | SET_CLOCK |
| Location | THOR range |

This command sets the THOR's battery backed clock to the current system time (set with SDATE).

**NOTE**

This is not actually necessary on the THOR XVI as this automatically alters the battery backed clock when the system clock is altered with SDATE or ADATE.

**CROSS-REFERENCE**

*SDATE* and *ADATE* alter the system clock. *A_SDATE* is similar on the ST/QL Emulator.

## 26.68 SET_FBKDT

| | |
|---|---|
| Syntax | SET_FBKDT #channel [,time] or<br>SET_FBKDT \file [,time] |
| Location | Level-2 drivers |

The command SET_FBKDT sets the date when a file was last backed-up. The time specified, must be in the number of seconds since 1st January 1961, ie. the number returned by DATE. If time is not specified or is 0, then the current DATE setting is used. If time=1 this has no effect on the file. Normally the backup date is not set unless you do so using SET_FBKDT. This command supports the data default directory (set with DATA_USE).

**Example**

```
SET_FBKDT \BOOT, DATE
```

sets the backup date on the file BOOT in the current data default directory to the current time and date.

**CROSS-REFERENCE**

*FBKDT*. See *FGETH$* for the structure of a file header, especially which byte is modified when the backup date is changed.

## 26.69 SET_FUPDT

| | |
|---|---|
| Syntax | SET_FUPDT #channel [,time] or<br>SET_FUPDT \file [,time] |
| Location | Level-2 drivers |

The command SET_FUPDT sets the date on which a file was last altered. This is always set to the current system DATE when a file is SAVEd, or CLOSEd after having been written to. If time is not specified (or is 0), then the current DATE is used. If time is set to 1, then this command will have no effect on the file. COPY sets the update time on the file being created to the current DATE. For a SuperBASIC 'backup' function which gives the newly created file the same update time as the original and alters the backup time, see FBKDT. This command supports the current default data directory (see DATAD$).

**NOTE**

If you use SET_FUPDT to alter the update time of a file OPENed to the specified channel, closing that channel later in the program will not affect the update time.

**CROSS-REFERENCE**

*FUPDT*. See *FGETH$* for the structure of a file header, especially which byte is modified when the update time is set.

## 26.70 SET_FVERS

| | |
|---|---|
| Syntax | SET_FVERS #channel [,version] or<br>SET_FVERS \file [,version] |
| Location | Level-2 drivers |

The command SET_FVERS sets the version number of a file - versions higher than 65535 or smaller than 0 are regarded as version MOD 65536, version=0 (or if version is omitted) means that the version number will not be updated when the channel to that file is closed. This command supports the current default data directory (see DATAD$).

**Example**

```
SET_FVERS \BOOT, 13
```

**CROSS-REFERENCE**

*FVERS*. See *FGETH$* for the structure of a file header, especially which byte is modified when the version is changed. The version number may be updated by *SAVE* and *QSAVE* on SMS.

## 26.71 SET_GREEN

| | |
|---|---|
| Syntax | SET_GREEN #channel, operation |
| Location | Windows (DIY Toolkit - Vol W) |

This command allows you to change the colours used within a specified window channel very quickly. In order to use this, you really need a good understanding of the way in which the QL display works - see the QL Display Appendix for some details. The effect that this command has on the specified window depends upon the value of operation:

| Operation | Effect |
|---|---|
| 0 | Clear all Green bits (remove any Green from the screen). |
| 1 | Set all Green bits. |
| -1 | If the Red bit for a pixel is set, Set the Green bit, otherwise clear it. |

**NOTE 1**

This command will only work on screen resolutions of 512x256 pixels.

**NOTE 2**

This command should not really be used in MODE 8.

**CROSS-REFERENCE**

*SET_RED* is similar. See *RECOL*. *W_SWAP* can also be used to recolour a window. Refer to the QL Display Appendix.

## 26.72 SET_RED

| | |
|---|---|
| Syntax | SET_RED #channel, operation |
| Location | Windows (DIY Toolkit - Vol W) |

This command is similar to SET_GREEN - the only difference is that instead of affecting green bits, it alters the red bits. The effect that this command has on the specified window depends upon the value of operation:

| Operation | Effect |
|---|---|
| 0 | Clear all Red bits (remove any Red from the screen). |
| 1 | Set all Red bits. |
| -1 | If the Green bit for a pixel is set, Set the Red bit, otherwise clear it. |

**CROSS-REFERENCE**

See *SET_GREEN* !

## 26.73 SET_LANGUAGE

| | |
|---|---|
| Syntax | SET_LANGUAGE country$ or<br>SET_LANGUAGE [country$] (THOR XVI v6.41 only) |
| Location | THOR range |

The command SET_LANGUAGE takes a string or name parameter and attempts to change the keyboard layout to the first one with a name of which the given parameter is an abbreviation (this comparison is case-independent). If the parameter is an empty string (or not specified), the next keyboard layout is selected. Ideally, in a program, the full name of the layout would be used for clarity. The search is circular which means that for example, if SET_LANGUAGE d was used, the Danish (Dansk) keyboard layout would be adopted rather than the German (Deutsch) layout, unless the Danish layout was already selected. The current keyboard layouts are supported:

| Number | Country$ | Language |
|---|---|---|
| 1 | International | None specific |
| 2 | British | English |
| 3 | Dansk | Danish |
| 4 | Deutsch | German |
| 5 | Espanol | Spanish (v4.20+ only) |
| 6 | Français | French |
| 7 | HELLAS | Greek |
| 8 | Suisse | Swiss |
| 9 | Svensk | Swedish (v4.20+ only) |

**Examples**

```
SET_LANGUAGE ""
```

jump to next keyboard layout in list.

```
SET_LANGUAGE 'Esp'
```

set layout to Spanish layout.

**NOTE 1**

Connected with each keyboard layout, there is also a national translation table, which you will need to install by using the command TRA 1.

**NOTE 2**

On THOR's equipped with a JS ROM, Français must be enclosed in quotation marks as it is an invalid variable name.

**NOTE 3**

The second variant of the command should not really be used as it is only supported on the v6.41 ROM for the THOR XVI. This has the same effect as:

```
SET_LANGUAGE "".
```

**CROSS-REFERENCE**

*LANGUAGE$* returns the name of the current keyboard layout in use. Before v6.41 of the THOR XVI, the keys <ALT><SYSREQ> had the same effect as *SET_LANGUAGE*. On v6.41, this keying was altered to call a Job called Alt_SysReq (Case dependent). *LANG_USE* allows SMS to use different languages for messages and errors. See also *KBD_TABLE*.

## 26.74 SET_XINC

| | |
|---|---|
| Syntax | SET_XINC #channel, increment |
| Location | DJToolkit 1.16 |

See *SET_YINC*, below, for details.

## 26.75 SET_YINC

| | |
|---|---|
| Syntax | SET_YINC #channel, increment |
| Location | DJToolkit 1.16 |

These two functions change the spacing between characters horizontally, *SET_XINC*, or vertically, SET_YINC. This allows slightly more information to be displayed on the screen. *SET_XINC* allows adjacent characters on a line of the screen to be positioned closer or further apart as desired. SET_YINC varies the spacing between the current line of characters and the next.

By choosing silly values, you can have a real messy screen, but try experimenting with *OVER* as well to see what happens. Use of the *MODE* or *CSIZE* commands in SuperBasic will overwrite your new values.

**EXAMPLE**

```
SET_XINC #2, 22
SET_YINC #2, 16
PRINT #2, "This is a line of text"
PRINT #2, "This is another line of text"
PRINT #2, "This is yet another!"
```

**CROSS-REFERENCE**

*SET_XINC*.

## 26.76 SEXEC

| | |
|---|---|
| Syntax | SEXEC device_file,start,length,data or<br>SEXEC device_file,start[,length[,data[,extra[,type]]] (Minerva v1.80+) or<br>SEXEC [device_]file,start,length,data (Toolkit II) or<br>SEXEC #channel,start,length,data (SMS only) |
| Location | QL ROM, Toolkit II |

In order for a program to be stored as an executable Job, it is necessary to store the machine code in a specified format on disk. The command SEXEC allows you to do this, taking a specified amount of code from memory and storing it in the specified file in a form which can later be EXECuted.

You will need to specify the start address of the machine code, the length of the code to be stored and the amount of data space to be given to the program when it is loaded back into memory (the data space represents the amount of working memory which is linked with the program when it is loaded, either for storing data at the end of the program or for the user stack - see a good QL machine code book for more details). The specified file name must include the name of the device to be used, unless Toolkit II is present, in which case the default program device is supported. If Toolkit II is present and the file already exists, you will be given the option of overwriting the file.

**Example**

To amend a given executable program, you may need to do the following:

```
100 length=FLEN(\example_exe)
110 datasp=FDAT(\example_exe)
120 start=RESPR(length)
130 LBYTES example_exe, start
140 POKE start + 1024, 100
150 SEXEC flp1_example_exe, start, length, datasp
```

**NOTE 1**

On Minerva ROMs (pre v1.80), if SEXEC was aborted for some reason whilst writing to a file, the file would be deleted. On later versions of Minerva and all other QL ROMs, the incomplete file is kept. Toolkit II reports Medium Full if this is the case.

**NOTE 2**

The Minerva variant is overwritten by the Toolkit II version of this command.

**NOTE 3**

On Minerva pre v1.83, SEXEC set the wrong file type!

**MINERVA NOTE**

On Minerva v1.80 (or later) the command SEXEC is practically the same as SBYTES. The only difference is the type parameter which defaults to 1 as opposed to 0 with SBYTES.

**SMS NOTE**

The fourth variant of this command allows you to save the data to an already existing channel which is OPEN to a file, thus cutting down on the number of times you need to access the file for error trapping (for example). See SBYTES for an example.

**WARNING**

Saving part of the QL's memory with SEXEC does not make it into EXECutable code - you must ensure that the program concerned has a proper Job header and conforms with the normal QDOS rules for EXECutable programs.

**CROSS-REFERENCE**

*SEXEC_O* is very similar. *EXEC* and *EXEC_W* allow you to load a program saved with *SEXEC*.

## 26.77 SEXEC_O

| | |
|---|---|
| Syntax | SEXEC_O [device_]file,start,length(Toolkit II) or<br>SEXEC_O device_file,start,length (THOR XVI) or<br>SEXEC_O #channel,start,length (SMS only) |
| Location | Toolkit II, THOR XVI |

This command is exactly the same as SEXEC except that it will automatically overwrite an existing file of the same name.

**NOTE**

The Toolkit II version of SEXEC_O supports the default data device.

**CROSS-REFERENCE**

See *SEXEC*.

## 26.78 SGN

| | |
|---|---|
| Syntax | SGN (x) and<br>SGN% (x) |
| Location | Math Package (SGN) and SGN (SGN%) |

Both functions work identically and return the sign of any valid number. The sign is defined as 1 for positive numbers, -1 for negative and 0 if the number is zero. Any number is allowed as a parameter.

**CROSS-REFERENCE**

*SIGN* is the same.

## 26.79 SGN%

See *SGN* above.

## 26.80 SHOOT

| | |
|---|---|
| Syntax | SHOOT |
| Location | ST/QL, QSound |

This command produces the sound of single gun shot.

**CROSS-REFERENCE**

*SND_EXT*, *BELL*, *EXPLODE*.

## 26.81 SI

| | |
|---|---|
| Syntax | SI |
| Location | Beuletools |

This function contains the control codes needed to switch on condensed print on an EPSON compatible printer:

```
PRINT SI
```

is the same as:

```
PRINT CHR$(15)
```

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*, *DBL*, *ENL*, *PRO*, *NRM*, *UNL*, *ALT*, *ESC*, *FF*, *LMAR*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 26.82 SIGN

| | |
|---|---|
| Syntax | SIGN (x) |
| Location | BTool |

See :ref`sgn`.

## 26.83 SIN

| | |
|---|---|
| Syntax | SIN (angle) |
| Location | QL ROM |

This function returns the sine of the given angle (in radians ranging from -PI/2 to PI/2). In a right-angled triangle, the sine is the ratio of the length of the side opposite to the angle, to the length of the hypotenuse. A negative angle indicates that the hypotenuse appears below the base line.

**Example**

A procedure to draw a sector of a circle with the centre at x,y and radius r.

- A is the angle between the first straight side of the sector and a vertical line on the screen,
- B is the angle between the two straight sides.
- Both angles have to be given in radians, b should be between 0 and 2*PI.
- Ch specifies the window to be used and cannot be omitted.

```
100 DEFine PROCedure SECTOR (ch, x, y, r, a, b)
110   LOCal x1, x2, y1, y2
120   x1 = x + r * SIN(a): x2 = x + r *SIN(a + b)
130   y1 = y + r * COS(a): y2 = y + r *COS(a + b)
```

(continues on next page)

(continued from previous page)

```
140   LINE# ch, x1, y1 TO x, y TO x2, y2
150   ARC# ch, x2, y2 TO x1, y1 ,b
160 END DEFine SECTOR

SECTOR #1, 50, 50, 10, PI/4, PI/2
```

**NOTE 1**

SIN (PI)==0 (approximately zero) on all ROMs. This should in fact equal zero - only the Lightning maths package and SMS get this right.

**NOTE 2**

On Minerva v1.96+ SIN with very large values for radian return 0. On other implementations it returns an overflow error. You should therefore check the range of the angle parameter.

**CROSS-REFERENCE**

See *COS*, *TAN*, *RAD*, *ASIN*, *ACOS*, *ATAN*, *DEG*. See also *SINH*. Please also refer to the Mathematics section of the Appendix.

## 26.84 SINH

| | |
|---|---|
| Syntax | SINH (x) |
| Location | Hyper, Hyperbola |

This function returns the hyperbolic sine (sinus hyperbolicus). The function is equivalent to: ( EXP(x) - EXP(-x) ) / 2 where the angle x (in fact a ratio) is a small floating point value.

**Example**

Draw a hyperbola and its asymptotes:

```
100 SCALE 10, -7, -5: PAPER 0: CLS: INK 3
110 LINE -4, -4 TO 4, 4, -4, 4 TO 4, -4: INK 7
120 FOR t = -2 TO 2 STEP 2E-2
130 x = COSH(t): y = SINH(t)
140 POINT x, y, -x, y
150 END FOR t
```

**CROSS-REFERENCE**

*COSH*, *TANH*, *ARSINH*

## 26.85 SINT

| | |
|---|---|
| Syntax | SINT (x) where x=0..65535 |
| Location | BTool |

The range of SuperBASIC integers is -32768 to 32767 - these are called signed integers because they can be negative. This compares to unsigned integers which have a different range, from 0 to 65535. The

function SINT converts unsigned integers to signed integers, which is not a very difficult task apart from the need to check the valid range:

```
signed% = unsigned - 2^16
```

or:

```
signed% = SINT(unsigned)
```

**CROSS-REFERENCE**

*UINT* converts in the other direction.

## 26.86 SIZE

| | |
|---|---|
| Syntax | SIZE ( array [{% \| $}] ) or<br>SIZE ( variable [{% \| $}] ) or<br>SIZE ( value) |
| Location | Math Package |

The function SIZE can take any kind of variable, array or constant. The returned value depends very much on the type of parameter: If a simple variable was passed, the function returns either 0 or 1, 1 if the variable points to any value or 0 if it does not, ie. if PRINT variable would show an asterisk to show that variable is not yet defined. Note that even though on SMS an unset variable does not show an asterisk when you use PRINT variable, this does not prevent this function from returning the correct value.

The return for a constant parameter such as:

```
PRINT SIZE(-22.3)
```

or:

```
PRINT SIZE("QL")
```

is always 1.

The return for arrays is entirely different. Passing an array tells SIZE to count its elements. Note the existence of a zero element, for example:

```
DIM a(2,2)
```

gives a nine elements in all:

```
a(0,0) a(0,1) a(0,2)
a(1,0) a(1,1) a(1,2)
a(2,0) a(2,1) a(2,2)
```

SIZE handles string arrays differently in that it returns the number of strings, not the number of characters, eg. for DIM a$(2,2), SIZE(a$) will not give 3*3 = 9 but 3.

Generally the return value of SIZE does not depend on the actual contents of the passed object. SIZE recognises if part of an object (especially strings and arrays) was passed.

**Examples**

```
DIM numbers(1,2,3,4,5)
PRINT SIZE(numbers)
```

returns 2*3*4*5*6=720.

```
yippie$="what a wonderful world"
PRINT SIZE(yippie$)
```

returns 1.

```
CLEAR PRINT SIZE(eeek)
```

returns 0

```
DIM string$(12,7,10)
PRINT SIZE(string$)
```

returns 13*8=104

```
PRINT SIZE(string$(1 TO))
```

returns 12*8=96.

**NOTE 1**

String arrays also contain numeric values - the first element (which is character zero) of a string contains the size of the string. For instance, take the above string$ array and then enter:

```
string$(4,4) = "knocking"
```

Now:

```
PRINT string$(4,4)
```

and you will see 'knocking' in #1.

```
PRINT string$(4,4,5)
```

gives the fifth character of knocking, the k, and:

```
PRINT string$(4,4,1)
```

the first one, again a k. And:

```
PRINT string$(4,4,0)
```

There is no character before the first, instead you will get the integer number 8 because:

```
LEN(string$(4,4))=8
```

This is tricky and not really necessary to know about as you can use LEN… just in case you come across the phenomenon and have wondered about it. See also DIM for a further explanation of strings.

**NOTE 2**

Before v2.06, this function may refuse to work on some implementations, giving 'Bad Parameter' error or returning the wrong value for string arrays.

**NOTE 3**

If the parameter is a single dimension string array, for example:

```
DIM a$(10)
PRINT SIZE (a$)
```

the value returned is 0. It is hoped that this will be fixed in a future version so that the value returned is 1.

**CROSS-REFERENCE**

*DIMN* and *NDIM* return other information about an array, eg: *PRINT* SIZE(a$) * DIMN(a$,NDIM(a$)) gives the total number of characters which can be stored in a string array a$. *LEN* returns the length of a string. *FREE_MEM* allows you to check how much memory an array uses.

## 26.87 SJOB

| | |
|---|---|
| Syntax | SJOB jobnr,timeout or<br>SJOB jobname,timeout |
| Location | TinyToolkit |

There are three ways in which a job can be made to do nothing:

1. Remove the job;
2. Set the job's priority to 0;
3. Suspend the job.

This command suspends the specified job for a specified period of time, which can be identified either by its jobnr (see JOBS) or by -1 (meaning the current job) or by its name (which need not be in quotes). Although suspending a job does not alter its priority, a suspended job will have no effect upon the speed of the QL. A positive timeout will stop the Job for timeout/50 seconds, whereas any negative number will suspend the job forever (ie. it can only be re-activated by an express command such as REL_JOB). The highest positive timeout is 32768 frames which is approximately 9 minutes, 6 seconds.

**Example 1**

```
SJOB "Quill", -1
```

will suspend Quill indefinitely.

```
SJOB Quill,-1
```

is the same even if there is a variable called Quill.

```
SJOB -1, 100
```

will suspend the current job for approx. 2 seconds.

```
SJOB 10, 100
```

will suspend Job number 10 for approx. 2 seconds

**Example 2**

A background Job which carries out work which is not time consuming, should not slow the whole system down, otherwise it is a complete waste of the computer's available time. Unfortunately, a priority of 1 is too high for a simple action such as checking the clock or updating key macros (See ALTKEY).

SJOB is useful to slow this job down to the desired speed. SJOB is also useful for setting PAUSEs independently of the machine's speed. The following program demonstrates both uses of SJOB and has to be compiled and executed as a multitasking job (ie. EXEC).

The priority of the job does not really matter, because the job only wakes up once a minute, looks at the clock and then drops off again.

```
100 REPeat Tower
110   d$=DATE$: minute=d$(16 TO 17)
120   SELect ON minute
130     =30:BEEP 20000,0,100,1000,0
140     =0:hour=d$(13 TO 14) MOD 12:IF hour=0:hour=12
150         FOR h=1 to hour: BEEP 10000,h,10,100,1: SJOB Q_MYJOB,65
160     =15:BEEP 5000,0,10,20,5000
170   END SELect
180   SJOB Q_MYJOB,3000
190 END REPeat Tower
```

This example needs Qliberator's Q_MYJOB function.

**NOTE**

As from v1.11, jobnr can be -1, so in the above example 2, you could use SJOB -1,65 and SJOB -1,3000 instead of the similar commands in lines 150 and 180 respectively. Earlier versions would also not accept a variable as the parameter for the job number.

**CROSS-REFERENCE**

*REL_JOB* releases a suspended job. *JOBS* lists all current jobs. *SUSJOB* and *TTSUS* are almost the same as *SJOB*.

## 26.88 SLOAD

| | |
|---|---|
| Syntax | SLOAD adr |
| Location | Ecran Manager |

This command takes part of a screen which has been saved with SSAVE and copies it to the visible screen, removing it from memory. SLOAD works like SSHOW with the sole difference that it can only be called once.

**NOTE**

This has the same problems as SSAVE.

**CROSS-REFERENCE**

*SSHOW*

## 26.89 SLUG

| | |
|---|---|
| Syntax | SLUG msecs<br>SLUG num (Q68) |
| Location | Gold Card (v2.24+), SMS, SMSQ/E for Q68 |

A disadvantage of the speed improvements by Gold Card (and later expansion boards) is that most games become simply too fast. The command SLUG can slow down the whole system by advising the operating system to read the keyboard less often (other solutions install background interrupts but some games suspend these). The parameter specifies the delay in milliseconds. The higher msecs, the slower the general operating speed will be. SLUG 5 to SLUG 10 on a Gold Card gives roughly the speed of a normal QL, but this depends very much on the software. Programs which do not spend a lot of time waiting for keyboard input such as interactive games, will not slow down so much. Only keyboard access is slowed down.

For the Q68 num must be in the range 0…255

**Example**

```
100 FOR n=0 to 1000 STEP 10
110   SLUG n
120   PRINT n
130   dummy=KEYROW(0)
140 END FOR n
```

**Note 1**

Since SLUG only slows down keyboard access (this is especially designed for arcade games), the above example would not be affected without line 130. All other lines run at maximum speed; the advantage is that screen output, which is a limiting factor for arcade games, is not affected by SLUG.

**Note 2**

SLUG for the Q68 is for the same purpose but operates in a different manner from SLUG on the Gold Card so line 130 is not actually necessary.

**CROSS-REFERENCE**

*SCR2DIS* and *CACHE_ON* can be used to speed up the computer's speed.

## 26.90 SMOVE

| | |
|---|---|
| Syntax | SMOVE scrno, adr [,xpos, ypos] |
| Location | Ecran Manager |

The command SMOVE will copy a stored screen (saved with SSAVE, where adr comes from) to the first (scrno=0) or second screen (scrno=1) - the latter is only possible if your system supports a dual screen mode.

Optionally, it is possible to specify a location where the screen part's upper left corner (absolute coordinates) should be placed; SMOVE will correct the xpos and ypos automatically if the restored picture would exceed the screen borders.

**NOTE**

See SSAVE.

**CROSS-REFERENCE**

*SSHOW*, *SLOAD*

## 26.91 SND_EXT

| | |
|---|---|
| Syntax | SND_EXT |
| Location | ATARI_REXT (v1.24 to v2.15) |

The ST-QL Emulators contain new extensions (based upon the QSound device) to enable programs to use the ST's sound facilities. Unfortunately, these extensions clash with the Turbo SuperBASIC compiler from Digital Precision. When the Emulator is started up, these sound extensions are switched off. SND_EXT will switch them back on. This command was replaced in v2.15 by ATARI_EXT.

You can test if the QSOUND interface (or these commands) are present by using:

```
PEEK_L(!! HEX('164'))
```

which will be 0 unless the commands are present (Turbo may also alter this figure whilst it is compiling a program).

**WARNING**

The sound extensions may crash the hardware.

**CROSS-REFERENCE**

Some of the available extensions for sound are *PLAY*, *RELEASE*, *BELL*, *SHOOT*, *EXPLODE*.

## 26.92 SNET

| | |
|---|---|
| Syntax | SNET no |
| Location | SMSQ/E, ATARI Emulators |

This command is similar to the NET command in that it sets the Network Station number of the machine on which it is issued. The only difference is that here it sets the station number for the SERNET Network (as opposed to the QNet Network).

**CROSS-REFERENCE**

See *SNET%*, *SNET_USE* and *NET*. Also please see *SERNET*, *MIDINET* and *FSERVE*.

## 26.93 SNET%

| | |
|---|---|
| Syntax | SNET% |
| Location | SMSQ/E, ATARI Emulators |

This function returns the current station number of the computer as set with SNET .

**CROSS-REFERENCE**

See *SNET*. *NET_ID* is similar.

## 26.94 SNET_ROPEN

| | |
|---|---|
| Syntax | SNET_ROPEN |
| Location | SMSQ/E, ATARI Emulators |

This command reopens the serial ports for use by the SERNET driver in case they have been closed by other programs.

**CROSS-REFERENCE**

See *SERNET*.

## 26.95 SNET_S%

| | |
|---|---|
| Syntax | SNET_S% (station) |
| Location | SMSQ/E, ATARI Emulators |

This function enables you to check whether a machine with the specified station number is connected to the SERNET . This can be useful to prevent the problem of the Network re-trying several times before failing when asked to send or read data from a Network station which does not exist.

**CROSS-REFERENCE**

See *SNET*.

## 26.96 SNET_USE

| | |
|---|---|
| Syntax | SNET_USE id |
| Location | SMSQ/E, ATARI Emulators |

Due to the fact that SERNET Networks can be run on computers alongside MIDINET Networks and even QNET Networks, it may be necessary to alter the identification letter used to access facilties on other computers in the Network. The default letter id is s, but this can be set to any other single letter by using this command. However, you should avoid letters which already appear as the first letter in another device driver (see DEVLIST).

**Example**

```
SNET_USE c
DEV_USE 3,c2_win1_
```

Redefine DEV3_ so that it refers to win1_ on station number 2 in the SERNET Network. This can be useful to allow some programs to access data over the Network. However note the file protection implemented in SERNET and MIDINET.

**NOTE**

Before v2.28 of Toolkit II, the various wildcard commands did not accept any single letter other than n as representing a Network.

**CROSS-REFERENCE**

See *SNET* and *SERNET*. Refer also to *SNET_S%*. *MNET_USE* is similar. See also *NFS_USE*.

## 26.97 SORT

| | |
|---|---|
| Syntax | SORT array$, offset [,row] |
| Location | ARRAY |

The SORT command takes a two or three-dimensional string array and sorts it in ascending order. offset is an even number which allows you to apply different sort criteria by telling SORT to compare the sub-strings to the right of position offset+1. The third, optional parameter is only necessary for three-dimensional arrays: it selects the row to be sorted.

**Example**

CAT lists a sorted directory, including deleted files, to window #1. Sorting the directory in fact requires just one line here (390), the entries are sorted by file length because the format of each entry is as follows:

```
1        10       20        30      37      45
|--------|--------|---------|-------|-------|
filename                            length
```

Changing the SORT in line 390 to:

```
390 SORT entry$, 0
```

will sort the list alphabetically. The other parts of the example PROCedure are written to require only Toolkit II, that makes reading the directory (the j loop from line 240 to 280) quite slow. If you are wondering why the file header is stored twice, both as a string (header$) and for direct memory access (header), this is for getting the best out of basic QL facilities, namely PEEK_W, PEEK_L and string slicing (line 310).

```
100 DEFine PROCedure CAT (dir$)
110   LOCal ch%, entries%, header, header$(64)
120   LOCal c%, l%, i
130   PRINT "Directory of"!dir$;": ";
140   ch% = FOP_DIR(dir$)
150   IF ch% < 0 THEN
160     PRINT \"Cannot open directory,"\"because ";
```

(continues on next page)

(continued from previous page)

```
170     REPORT#1, ch%: RETurn
180   END IF
190   entries% = FLEN(#ch%) / 64
200   DIM entry$(entries%, 45)
210   header = ALCHP(64)
220   FOR i = 0 TO entries% - 1
230     header$ = ""
240     FOR j = 0 TO 63
250       BGET#ch%, c%
260       POKE header+j, c%
270       header$ = header$ & CHR$(c%)
280     END FOR j
290     l% = PEEK_W(header + 14)
300     IF l% THEN
310       entry$(i) = header$(17 TO 16 + l%) & FILL$(" ", 37 - l%)
320       entry$(i) = entry$(i) & (PEEK_L(header) - 64)
330     ELSE
340       entry$(i) = "(deleted)" & FILL$(" ", 28) & "n.a."
350     END IF
360     PRINT ".";
370   END FOR i
380   CLOSE#ch%: RECHP header: PRINT
390   SORT entry$, 36
400   FOR i = 0 TO entries% - 1
410     PRINT entry$(i)
420   END FOR i
430 END DEFine CAT
```

**CROSS-REFERENCE**

*SEARCH* searches string arrays.

## 26.98 SOUNDEX

| | |
|---|---|
| Syntax | SOUNDEX (word$) |
| Location | Ähnlichkeiten |

This function returns an integer which represents the word contained in the string passed as a parameter, in such a way that for two English words which sound similar, the same results are returned. Internally, each character is replaced by a cipher and then all double (triple etc) ciphers are removed.

**Examples**

```
SOUNDEX ("user"): REMark 26
SOUNDEX ("looser"): REMark 426
SOUNDEX ("l'user"): REMark 426
```

**NOTE**

The difference between two SOUNDEX results is not proportional to the phonetic difference between the parameters.

**CROSS-REFERENCE**

*WLD* calculates such a difference, *PHONEM* is similar to *SOUNDEX*.

## 26.99 SPJOB

| | |
|---|---|
| Syntax | SPJOB jobname,priority (Toolkit II, TinyToolkit pre v1.10 and THOR only) or<br>SPJOB jobnr,tag,priority (Toolkit II and THOR only) or<br>SPJOB jobID,priority or<br>SPJOB jobnr,priority(TinyToolkit pre v1.10) |
| Location | Toolkit II, THOR XVI, TinyToolkit (pre v1.10), BTool |

The specified job (described by either its jobname, its job number and tag, or its job identification number) is set to the given priority (which should be in the range 0 to 127 to maintain compatibility with Minerva). A priority of zero will ensure that the job waits until it is given a higher priority by another job.

**NOTE 1**

It is possible that only the second syntax works. Get an update!

**NOTE 2**

Before v1.10 of TinyToolkit, this toolkit included the same command but with an incompatible syntax - this version has been renamed SP_JOB.

**MINERVA NOTES**

Although on other ROMs, a priority higher than 127 can be assigned to a job, on Minerva, the permitted priority range is actually -128…127 (if a priority is stated to be higher than 127, you must subtract the difference between this number and 256 from 0 to get the negative priority).

The idea behind these negative priorities is that they are for 'background tasks' which will only run when no tasks with a positive priority are running. However, the effect is slightly more complex because these negative priorities are split into eight levels, each of which can have jobs running with priorities equivalent to -1 to -15. A job in one level will not run whilst a job in a higher level is running, however within each level each job will get a different amount of processor time depending on their priorities {a job with a lower priority (eg. -15) will get more processing time than a job with a higher priority (eg. -1)}.

| Level | Priority Range | Overall Value |
|---|---|---|
| 0 | -1 … -15 | -1 … -15 |
| 1 | -1 … -15 | -16 … -31 |
| 2 | -1 … -15 | -32 … -47 |
| 3 | -1 … -15 | -48 … -63 |
| 4 | -1 … -15 | -64 … -79 |
| 5 | -1 … -15 | -80 … -95 |
| 6 | -1 … -15 | -96 … -111 |
| 7 | -1 … -15 | -112 … -127 |

**WARNING**

The supplied parameters are not checked to see what you are trying to do, which means that you can use this command to set the priority of SuperBASIC to zero, preventing further command entry.

**CROSS-REFERENCE**

*SJOB* suspends a job, *REL_JOB* releases it. *RJOB* and *KJOB* remove a specific job, *KILL* and *KJOBS* remove all jobs except the main SuperBASIC interpreter. See also *SP_JOB*, *PRIO*, *PRIORITISE*.

## 26.100 SPL

| | |
|---|---|
| Syntax | SPL {input \| #ch} [TO {output \| #ch}] |
| Location | Toolkit II, THOR XVI |

It can sometimes be useful to copy a file in the background. The command SPL sets up a small Job which runs at a low priority and acts as a print spooler, reading the whole of the input data from the given input device as quickly as possible and then just outputting the data when it can. Although control is returned to the calling program quite quickly, both the input and output files are left open until SPL has completed its job.

SPL is mainly for outputting files to a printer in the background (allowing you to carry on other work in the meantime).

If however, a file is specified as the output, the SPL command acts like COPY_O, except in the background. If output is not specified, the SPL command uses the default destination device. Existing channel numbers may also be specified as the input and output names, provided that both channels are already open for input and output respectively.

**Examples**

```
SPL flp1_Example_txt TO SER
```

prints the file flp1_Example_txt in the background.

```
SPL_USE SER:
SPL flp1_Example_txt
```

this is the same as example 1.

**WARNING**

If the default destination device is a directory device and you do not specify a file for output, the SPL job may never complete its task and leave files open.

**CROSS-REFERENCE**

See *COPY_O* and *SPLF*. *SPL_USE* and *DEST_USE* allow you to alter the default destination device.

## 26.101 SPLF

| | |
|---|---|
| Syntax | SPLF {input \| #ch} [TO {output \| #ch}] |
| Location | Toolkit II, THOR XVI |

This is exactly the same as SPL except that at the end of sending the output, a form feed symbol, CHR$(12) is sent. SPLF is obviously intended for use with printers.

**CROSS-REFERENCE**

See *SPL*.

## 26.102 SPL_USE

| | |
|---|---|
| Syntax | SPL_USE name |
| Location | Toolkit II, THOR XVI |

This command sets the default destination device and therefore has a similar effect to DEST_USE. However, this command is slightly improved, in that if the supplied name does not end in an underscore, this is taken to be an external device port (such as SER) and no underscore is added.

**Examples**

```
DEST_USE flp2_Quill: COPY ram2_Letter_doc
```

will copy the file ram2_letter_doc to flp2_Quill_letter_doc.

```
SPL_USE ser: COPY ram2_Letter_txt
```

will copy the file ram2_Letter_txt to the serial port, ser.

**NOTE**

SPL_USE will overwrite the default destination device set with DEST_USE.

**CROSS-REFERENCE**

*DESTD$* returns the current default destination device. Also see *PROG_USE*, *DLIST*, *DATA_USE*, *DEST_USE*, *DDOWN*, *DUP*, and *DNEXT*.

## 26.103 SP_JOB

| | |
|---|---|
| Syntax | SP_JOB jobname, priority or SP_JOB jobnr, priority |
| Location | TinyToolkit (v1.10+) |

Acts just like SPJOB.

**NOTE**

As from v1.11, the jobnr may be -1 to mean the current job. Earlier versions would not allow jobnr to be a variable either.

**CROSS-REFERENCE**

See *SPJOB*.

*JBASE* contains details of the different parameters jobname and jobnr.

## 26.104 SQR

| | |
|---|---|
| Syntax | SQR (x) |
| Location | Math Package |

See SQRT below!

## 26.105 SQRT

| | |
|---|---|
| Syntax | SQRT (x) |
| Location | QL ROM |

This function returns the square root of the given parameter. The opposite of this function is x^2. The given parameter can be zero or any positive value.

**Example**

```
PRINT SQRT(32768*2) will return 256.
```

**NOTE 1**

The version of SQRT implemented on Minerva v1.90 (or later) is the fastest version of this command which we have seen anywhere!

**NOTE 2**

On Minerva pre v1.96:

```
SQRT(4^x*(12^31))
```

was returning the negative square root. It now returns the positive square root.

**CROSS-REFERENCE**

*ABS* will return the absolute value of the given parameter.

## 26.106 SSAVE

| | |
|---|---|
| Syntax | SSAVE (scrno, xpos, ypos, xsiz, ysiz) |
| Location | Ecran Manager |

The function SSAVE reserves memory and saves a part of the screen to it, the saved block's left upper corner is the point (xpos,ypos) in absolute co-ordinates, the width is xsiz and the height ysiz. xpos may range from 0 to 511 and ypos from 0 to 255, so SSAVE is not suitable for resolutions other than 512x256 pixels. The reserved memory can only be released with SLOAD. The first parameter scrno can be either 0 or 1 - it is used under dual screen mode to select the first or second screen, scrno=1 is only available under dual screen mode, on other machines SSAVE will break with the 'not found' (-7) error. The value returned represents the address where the screen is stored in memory.

**NOTE**

If you wish to link the Ecran Manager Toolkit to a QLiberated program, you must not use the ECMAN but the ECMANcp version.

**CROSS-REFERENCE**

Saved pictures can be reloaded with *SSHOW*.

## 26.107 SSHOW

| | |
|---|---|
| Syntax | SSHOW adr |
| Location | Ecran Manager |

This command restores a screen part saved with SSAVE, therefore the parameter adr must be the value returned by the SSAVE function. The memory area where the picture is saved is unaffected, so SSHOW can be executed any number of times.

**NOTE**

See SSAVE.

**CROSS-REFERENCE**

*SLOAD* displays a saved screen part and frees the memory used, *SMOVE* allows you to view such a saved screen part at a different location or on a different screen.

## 26.108 SSTAT

| | |
|---|---|
| Syntax | SSTAT |
| Location | Ecran Manager |

The function SSTAT returns either 0 or 1, corresponding to the first or second screen. The function is used to find out which of these screens is currently the visible screen. Unless you have Minerva or Amiga QDOS set up in dual screen mode, this is always 0.

**Example**

Force the second screen to be displayed (this only works in Minerva or Amiga QDOS):

```
IF SSTAT = 0 THEN MODE 80,-1
```

**NOTE**

See SSAVE.

**CROSS-REFERENCE**

*DEFAULT_SCR*, *SCRON*, *SCROF*, *MODE*

## 26.109 SSTEP

| | |
|---|---|
| Syntax | SSTEP [{#ch \| device_file}] [; [first] [TO [last]]] |
| Location | Minerva (TRACE) |

Minerva is supplied with a very simple trace routine on the utility disk supplied with Minerva, stored in the file trace_bin. Before using the trace function, you will need to link in trace_bin with the line:

```
LRESPR flp1_trace_bin
```

or something similar. Having done this, you can turn on the tracing function with SSTEP which will print to the given channel (default #0) or file, each line number and statement just before it is performed in the format: line_no : statement_no.

You can also supply the trace function with a line range, so that it will only report on statements being executed within the given line range. The line range defaults to: 1 TO 32767.

Whilst the trace function is enabled and the program is running within the given range, the interpreter will wait for a key to be pressed between each statement. As each command in each statement is executed, a single character is shown by the trace routine to represent the type of the command to be executed. However, the meaning of these symbols has never been revealed. In single-step mode, you need to press a key between each command!!

**NOTE**

This trace toolkit will only work on Minerva.

**CROSS-REFERENCE**

See *TRON* and *TROFF*.

## 26.110 STAMP

| | |
|---|---|
| Syntax | STAMP string$ |
| Location | STAMP |

This command is the same as FORCE_TYPE !

## 26.111 STAT

| | |
|---|---|
| Syntax | STAT [#channel,] [device] or<br>STAT \file [,device] |
| Location | Toolkit II, THOR XVI |

This command prints the name of a medium inserted into the given device and the available sectors to the given channel (default #1), or file. The device must be a directory device, such as FLP1_ (but not PAR or CON). If no device is stated, then the default data device is used.

**Examples**

```
STAT STAT ram1_
STAT n2_win1_
STAT #3,flp2_
STAT #0 STAT \mdv2_
STAT _dat
STAT \ram5_TMP,mdv1_
```

**CROSS-REFERENCE**

*DLIST* shows the default devices, *DATAD$* holds the default data device. Change default devices with *DATA_USE*, *PROG_USE* and *SPL_USE*. *DIR* and *WSTAT* provide other information about directory devices.

## 26.112 STEP

| | |
|---|---|
| Syntax | ... STEP stepwidth |
| Location | QL ROM |

This keyword forms part of the FOR structure and has no meaning on its own. Any attempt to enter it on its own will result in a 'Bad Line' error.

**CROSS-REFERENCE**

See *FOR*!

## 26.113 STOP

| | |
|---|---|
| Syntax | STOP |
| Location | QL ROM, Toolkit II |

This command forces an interpreted program to be terminated at the position where STOP appears in the listing. The program can then be continued (provided that the message 'PROC/FN cleared' has not appeared) by using the command CONTINUE. Compiled programs terminate and remove themselves when STOP is encountered.

**Example**

This program will print 1 and 2 to channel #1 and stop at line 120. If it runs under the interpreter, CONTINUE will restart at line 130 (after STOP) and print 3 and 4. RETRY does not continue here because it tries to re-run line 120 and stops again.

```
100 PRINT 1
110 PRINT 2
120 STOP
130 PRINT 3
140 PRINT 4
```

**NOTE**

If Toolkit II is installed, STOP clears WHEN ERRor definitions.

**CROSS-REFERENCE**

*RUN* starts a program and *GO TO* jumps to a specified line. See *CONTINUE*, *RETRY*. Also see *QUIT*.

## 26.114 STRIP

| | |
|---|---|
| Syntax | STRIP [#ch,] colour |
| Location | QL ROM |

Whenever a character is printed to the QL screen, it is made up of two components - the character itself which appears in the current INK colour, and the rectangular block on which the character has been formed. The latter is known as the 'strip' of the character and the size of this strip depends on the current character size and spacing (see CSIZE).

Normally, when you set the PAPER colour of a window, the character STRIP is set to the same colour. However, you may wish to print characters on a different background colour in order to make them stand out. STRIP allows you to alter the colour of the character background in the specified window (default #1) to a given colour (or composite colour). However, if you want to print characters in a window without using this character background (ie. forming a transparent strip), you will need to use the commands OVER 0 or OVER -1 (see OVER for more details).

**Example**

A simple routine for printing out a Title on screen:

```
10 WINDOW 512, 256, 0, 0: PAPER 4
20 MODE 4: CLS
30 TITLE #1,'This is a Title', 120, 95
40 :
100 DEFine PROCedure TITLE(ch,text$,x,y)
110   CSIZE 2,1: OVER 0
120   CURSOR #ch,x-2,y+1
130   STRIP #ch,0: PRINT #ch,FILL$(' ',LEN(text$))
140   CURSOR #ch,x,y
150   STRIP #ch,2: INK #ch,7
160   PRINT #ch,text$
170   CURSOR #ch,x-2,y+1
180   OVER 1: INK#ch,0
190   PRINT #ch,text$
200 END DEFine
```

**NOTE**

The STRIP colour is automatically reset to the same as the PAPER colour following a PAPER command.

**CROSS-REFERENCE**

*PAPER* also sets the *STRIP* colour. Compare *IO_TRAP*. *CSIZE* and *CHAR_INC* allow you to alter the spacing between characters. *INK* contains details of standard and composite colours. See also *INVERSE* which can also prove useful.

## 26.115 SUB

| | |
|---|---|
| Syntax | ... SUB line |
| Location | QL ROM |

This keyword forms part of the SuperBASIC keyword GO SUB and has no purpose on its own. Any attempt to use it on its own will cause a 'Bad Line' error.

**CROSS-REFERENCE**

See *GO SUB*!

## 26.116 SOUNDFILE

| | |
|---|---|
| Syntax | SOUNDFILE filename$[,rep%] |
| Location | SMSQ/E for Q68 |

Loads and plays a sound file through the SMSQ/E Sampled Sound System. If rep% is present the file will be played rep%+1 times.

**Note 1**

The file is not loaded into memory so it should be played from a fast device like a ram disk.

**CROSS-REFERENCE**

*SOUNDFILE2*, *SOUNDFILE3*, *KILLSOUND*

## 26.117 SOUNDFILE2

| | |
|---|---|
| Syntax | SOUNDFILE2 filename$[,rep%] |
| Location | SMSQ/E for Q68 |

Loads and plays a sound file through the SMSQ/E Sampled Sound System. If rep% is present the file will be played rep%+1 times.

Just like SOUNDFILE except using another job owned by the caller so that the command returns immediately.

**Note 1**

The file is not loaded into memory so it should be played from a fast device like a ram disk.

**CROSS-REFERENCE**

*SOUNDFILE*, *SOUNDFILE3*, *KILLSOUND*

## 26.118 SOUNDFILE3

| | |
|---|---|
| Syntax | SOUNDFILE3 filename$[,rep%] |
| Location | SMSQ/E for Q68 |

Loads and plays a sound file through the SMSQ/E Sampled Sound System. If rep% is present the file will be played rep%+1 times.

Just like SOUNDFILE except using another job not owned by the caller so that the command returns immediately.

**Note 1**

The file is not loaded into memory so it should be played from a fast device like a ram disk.

**CROSS-REFERENCE**

*SOUNDFILE*, *SOUNDFILE2*, *KILLSOUND*

## 26.119 SUSJOB

| | |
|---|---|
| Syntax | SUSJOB jobId,timeout |
| Location | BTool |

See *SJOB*.

## 26.120 SWAP

| | |
|---|---|
| Syntax | SWAP var1,var2 or<br>SWAP var1$,var2$ |
| Location | SWAP, Math Package |

This command exchanges the values of the two variables. The parameters can be either numeric variables (integer and floating point) or strings. Arrays are not allowed and both variables have to be the same type: SWAP a$,b is illegal, even if a$ contained a valid number. Also, constant expressions such as SWAP a%,3 are not allowed, since this would not make any sense. Unfortunately, it is not possible to SWAP two elements of an array, the example shows why this would be practicable. The Math Package variant also allows you to swap whole arrays.

**Example**

In most kinds of sorting routines, a lot of swapping is necessary and an assembler routine which takes over this work makes the process quicker. Here is a Quicksort algorithm as a general subroutine. field$ is sorted from the left element to the right.

```
100 DEFine PROCedure QSort (field$,left,right)
110   LOCal i,j,last$
120   i=left: j=right: last$=field$(j)
130   REPeat SortLoop1
140     REPeat SortLoop2:IF field$(i)<last$:i=i+1:ELSE EXIT SortLoop2
150     REPeat SortLoop2:IF field$(j)>last$:j=j-1:ELSE EXIT SortLoop2
160     IF i<=j THEN
170       f1$=field$(i): f2$=field$(j): SWAP f1$,f2$
180       field$(i)=f1$: field$(j)=f2$
190       i=i+1: j=j-1
200     END IF
210     IF i>j THEN EXIT SortLoop1
220   END REPeat SortLoop1
230   IF left<j THEN QSort field$,left,j
240   IF right>i THEN QSort field$,i,right
250 END DEFine QSort
```

Compilers have a fixed stack size - you might have to raise this because this procedure iterates (ie. calls itself), which is something which eats up the stack very quickly. The SuperBASIC interpreter uses a flexible stack.

**CROSS-REFERENCE**

*LET*

## 26.121 SXTRAS

| | |
|---|---|
| Syntax | SXTRAS [#channel,] [character] |
| Location | TinyToolkit |

This command lists all machine code SuperBASIC extensions in alphabetic order to the given channel (default #1). If a character is specified, then only those commands which appear later alphabetically will be listed - if character is longer than one character, only the first character is recognised.

**Example**

SXTRAS s

**CROSS-REFERENCE**

*EXTRAS* and *TXTRAS* do not sort the keywords. Also look at *VOCAB*.

## 26.122 SYNCH%

| | |
|---|---|
| Syntax | SYNCH% |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I) |

This function is only used to debug the DIY Toolkit mouse drivers. It returns a value which is in the range 1…3 for a two button mouse and 1…5 for a three button mouse. The value indicates which byte of the mouse message is due next and therefore when being used, you should see it circling between the upper and lower limits of the range. However, if the byte is corrupt or lost, then the value returned by this function will be zero. This can indicate problems with either your serial port or the interface to the mouse - if the former, you should consider getting Hermes for your computer.

**CROSS-REFERENCE**

See *PTR_ON* and *PTR_FN%*.

## 26.123 SYSBASE

| | |
|---|---|
| Syntax | SYSBASE |
| Location | QBASE (DIY Toolkit - Vol Q) |

The function SYSBASE is identical to SYS_BASE, see below. Don't forget: never assume that the System Variables are located at 163840 ($28000). They can move!!

## 26.124 SYS_BASE

| | |
|---|---|
| Syntax | SYS_BASE |
| Location | SYSBASE, Fn |

The function SYS_BASE returns the base address of the system variables.

**Example**

```
POKE_W SYS_BASE+140,8
POKE_W SYS_BASE+142,3
```

Sets the key repeat delay.

**NOTE 1**

Users peeking and poking in the System Variables should know what they are doing!

**NOTE 2**

Minerva and SMS offer another technique to read and alter system variables but these are specific to Minerva and SMS whilst SYS_BASE works an every ROM. It is generally not advisable to access fixed addresses in memory as virtually everything can move around.

**CROSS-REFERENCE**

*SYSBASE*, *WIN_BASE*, *PEEK*, *POKE*, *SCREEN*, *SYS_VARS*, *VER$*

## 26.125 SYS_VARS

| | |
|---|---|
| Syntax | SYS_VARS |
| Location | THOR (all models) |

The function SYS_VARS returns the base address of the system variables, which can move around on the THOR range of computers, in much the same way as they can move on other implementations - it is therefore imperative that any program which uses the system variables works relative to the address returned by this function.

**Example**

```
POKE SYS_VARS+133,-1
```

switches off the THOR XVI's windowing facilities for windows opened after this command.

**CROSS-REFERENCE**

*VER$*(-2) on Minerva ROMs and on SMS returns the base address of the system variables, as do *SYS_BASE* and *SYSBASE*.

## 26.126 S_FONT

| | |
|---|---|
| Syntax | S_FONT [#channel,] font1, font2 |
| Location | FONTS |

This command is exactly the same as CHAR_USE.

**CROSS-REFERENCE**

See *CHAR_USE* and *CHAR_DEF*. See also the Appendix on Fonts.

## 26.127 S_LOAD

| | |
|---|---|
| Syntax | S_LOAD adr |
| Location | TinyToolkit |

S_LOAD takes an address (adr) returned by S_SAVE and displays the saved screen just like S_SHOW does. Additionally, the reserved memory to which adr points is released so that it can be used for other purposes. S_LOAD therefore only works once on a given address.

**NOTE 1**

Under odd conditions S_LOAD will load and show more than had been stored with S_SAVE. The Win… set of commands replace the S_… set and get around these problems.

**NOTE 2**

S_LOAD assumes that it needs to copy the stored screen to $20000 and that will not therefore work on Minerva's second screen. It also assumes the screen is 512x256 pixels and will not work on higher resolutions or under dual screen mode.

**WARNING**

A wrong address leads to crashes!

## 26.128 S_SAVE

| | |
|---|---|
| Syntax | S_SAVE (#wind) |
| Location | TinyToolkit |

This function causes the contents of the window #wind to be stored in memory and the address is then returned. Do not forget the return value! #wind must be a window or a bad parameter error (-15) is reported.

**Example**

```
100 CLS
110 PRINT PEEK$(0,1000)
120 adr1 = S_SAVE(#1)
130 CLS
```

(continues on next page)

(continued from previous page)

```
140 PRINT PEEK$(100,1000)
150 adr2 = S_SAVE(#1)
160 FOR n=1 TO 20: S_SHOW adr1: S_SHOW adr2
170 S_LOAD adr1: S_LOAD adr2
```

**NOTE**

Although S_SAVE will save a window stored on the second screen provided by Minerva and Amiga QDOS, it assumes the screen resolution is 512x256 pixels and cannot work with higher resolutions.

**CROSS-REFERENCE**

*S_LOAD* and *S_SHOW* view the saved screen part. Memory taken by *S_SAVE* cannot be freed with *RECHP* or *CLCHP*, only with *S_LOAD*. See also *SCR_STORE* and *SAVEPIC* for alternatives.

## 26.129 S_SHOW

| | |
|---|---|
| Syntax | S_SHOW adr |
| Location | TinyToolkit |

Adr must be a value returned by S_SAVE: the command S_SHOW displays the screen information stored by S_SAVE. The screen is however retained in memory for future access.

**NOTE**

This command suffers with the same problems as S_LOAD.

**WARNING**

A wrong address leads to serious crashes.

**CROSS-REFERENCE**

*S_SAVE*

## 26.130 SYSTEM_VARIABLES

| | |
|---|---|
| Syntax | sys_vars = SYSTEM_VARIABLES |
| Location | DJToolkit 1.16 |

This function returns the current address of the QL's system variables. For most purposes, this will be hex 28000, decimal 163840, but Minerva users will probably get a different value due to the double screen. *Do not* assume that all QLs, current or future, will have their system variables at a fixed point in memory, this need not be the case.

**EXAMPLE**

```
PRINT SYSTEM_VARIABLES
```

CHAPTER

# TWENTYSEVEN

# KEYWORDS T

## 27.1 TAN

| | |
|---|---|
| Syntax | TAN (angle) angle <> (2n+1) * PI/2 (n=0,1,2,...) |
| Location | QL ROM |

This function calculates the tangent of an angle given in radians. The solution of TAN(PI/2) is not actually defined because the definition of TAN is TAN(x)=SIN(x)/COS(x) and COS(PI/2)=0. In practice, most ROM implementations will return a value of about 1E10 instead of an error because they calculate COS(PI/2)<>0. Due to the periodic nature of this function function, values for angle should really be in the range -PI/2 < angle < PI/2.

**Example**

```
100 WINDOW 448,200,32,16: PAPER 3: CLS
110 SCALE 8,-.2,-.2: INK 7
120 INPUT "Angle (0..90):"!angle
130 INPUT "Speed (..11 m/s):"!speed
140 angle=RAD(angle): c1=TAN(angle)
150 c2=9.81 / 2 / speed^2 / COS(angle)^2
160 :
170 FOR x=0 TO c1/c2 STEP c1/c2/20
180   y=c1 * x - c2 * x^2
190   FILL 1: CIRCLE x,y,.2: FILL 0
200 END FOR x
```

**NOTE 1**

TAN(PI)==0 on all implementations - this should be zero. Only SMS currently corrects this.

**NOTE 2**

On Minerva v1.96+ large values of angle return 0. On other ROMs it produces an overflow error.

**CROSS-REFERENCE**

*SIN*, *COS*, *COT*, *ASIN*, *ACOS*, *ATAN* and *ACOT* are other common trigonometrical functions. *RAD* converts degrees into radians, *DEG* vice-versa. Please also refer to the Mathematics section of the Appendix.

## 27.2 TANH

| | |
|---|---|
| Syntax | TANH (x) |
| Location | Hyper, Hyperbola |

This function is analogous to the tangent (TAN) - the hyperbolic tangent (TANH) is the hyperbolic sine divided by the hyperbolic cosine:

```
TANH(x) = SINH(x) / COSH(x)
```

resulting in the following formula (if SINH and COSH are replaced by their definitions):

```
TANH(x) = (EXP(x) - EXP(-x)) / (EXP(x) + EXP(-x))
```

**CROSS-REFERENCE**

*ARTANH* is the inverse function of *TANH*, *COTH* a complementary function to *TANH*.

## 27.3 TCA

| | |
|---|---|
| Syntax | TCA (i,n) |
| Location | Toolfin |

The function TCA returns the value of: $i/(1-(1+i)^{-n})$ where i and n can be any floating point numbers (see MT for error handling).

**CROSS-REFERENCE**

*MT*, *VA*, *VFR*, *VAR*, *TNC*, *TEE*, *RAE*, *RAFE*

## 27.4 TCONNECT

| | |
|---|---|
| Syntax | TCONNECT #pipechan TO #anychan |
| Location | TinyToolkit |

Pipes are serial devices for buffered data transfer, they have two sides:

- The input pipe which puts any data into the buffer until it's full and;
- the output pipe which reads the data from the buffer:

```
input pipe ---> buffer ---> output pipe
```

There are two kinds of pipes on the QL:

- Standard pipes are part of the original QL ROM, the input pipe device name is pipe_<buffer> where <buffer> is the buffer size in bytes (1..32767). It is necessary to know the CHANID of the input pipe to open the output pipe, see FILE_OPEN.

- The second type are named pipes which have the same concept except that the output pipe can be identified by name: The input pipe is pipe_<name>_<buffer> and the output pipe pipe_<name>. See the Appendix on Device Drivers regarding Pipes for more information.

TCONNECT makes standard pipes useable: the command expects two opened channels where the first, #pipechan, must be an input pipe and the second, #anychan can be anything. TCONNECT changes the internal meaning of #anychan so that it becomes an output pipe connected to the input pipe #pipechan:

Before TCONNECT:

```
#pipechan -> input pipe -> buffer scr_2x2 <- #anychan
```

After TCONNECT:

```
#pipechan -> input pipe -> buffer -> output pipe -> #anychan
```

**Example**

DEVLIST$ returns the devices listed by DEVLIST in a string, separated by spaces. ISDEVICE takes a device and checks with the help of DEVLIST$ if it is a legal device:

```
100 DEFine FuNction DEVLIST$
110   LOCal list$,dev$: list$=""
120   OPEN#3,pipe_80
130   OPEN#4,scr_
140   TCONNECT #3 TO #4
150   DEVLIST#3
160   INPUT#4,dev$\dev$
170   REPeat read_devs
180     IF NOT PEND(#4) THEN EXIT read_devs
190     INPUT#4,dev$
200     list$=list$&" "&dev$
210   END REPeat read_devs
220   CLOSE#3: CLOSE#4
230   RETurn list$
240 END DEFine DEVLIST$
```

A legal drive device consists of three letters (the device name), a drive number (1..8) and an underscore:

```
250 :
260 DEFine FuNction ISDEVICE(dev$)
270   IF LEN(dev$)<>5 THEN RETurn 0
280   IF dev$(5)<>"_" THEN RETurn 0
290   IF dev$(4)<"1" OR dev$(4)>"8" THEN RETurn 0
300   IF NOT (dev$(1 TO 3) INSTR DEVLIST$) THEN RETurn 0
310   RETurn 1
320 END DEFine ISDEVICE
```

**CROSS-REFERENCE**

See *FILE_OPEN*, *CHANID*, pipes and especially *PEND*. Some more examples appear at *FILE_LEN* and *FUPDT*. *QLINK* is the same. Qliberator gives the QCONNECT command which is the same.

## 27.5 TEE

| | |
|---|---|
| Syntax | TEE (i,n) |
| Location | Toolfin |

The function TEE returns the value of $(1+(i/n)^n-1)$ where i and n can be any floating point numbers (see MT for error handling).

**CROSS-REFERENCE**

*MT*, *VA*, *VFR*, *VAR*, *TCA*, *TNC*, *RAE*, *RAFE*

## 27.6 THEN

| | |
|---|---|
| Syntax | THEN [statement] *[:statement]* |
| Location | QL ROM |

This keyword is optional and is used as part of the IF..END IF structure. It has no use on its own.

**CROSS-REFERENCE**

See *IF*.

## 27.7 THING

| | |
|---|---|
| Syntax | THING (thingname$) |
| Location | Fn |

This function is used to check whether a Thing is present in memory (ie. whether a given item appears on the Thing list). If the Thing is present, then the function will return 0, otherwise it will return -7 (not found).

**Examples**

```
PRINT THING('Button_sleep')
PRINT THING('HOTKEY')
```

**NOTE**

In versions earlier than v1.02, this function may sometimes return a value greater than zero if the Thing exists.

**CROSS-REFERENCE**

*TH_VER$* explains what Things are.

## 27.8 TH_FIX

| | |
|---|---|
| Syntax | TH_FIX |
| Location | SMSQ/E and ST/QL Level B-11 drivers onwards |

This command fixes some programs which were written before the current Thing List was standardised (in the Level B-10 drivers for the ST/QL Emulator) and allows them to work under current versions by adopting the old style Thing List. You should really update the problem program.

## 27.9 TH_VER$

| | |
|---|---|
| Syntax | TH_VER$ (thingname$) |
| Location | Fn |

Things in QDOS terms refer to an extension of QDOS which was introduced by the Thing System provided by Qjump's Extended Pointer Interface and was also implemented (although slightly differently) on the THOR XVI computer. It is an universal storage method for named resources.

A Thing List is created by the Thing System which lists all of these named resources, which can range from a piece of machine code to a printer driver (and much more). The idea is that any program which wants to access a specified utility or driver need only search in this list to see if the Thing is installed in the current system, and then pointers contained in this list allows the program to access the Thing (if available).

Each Thing can be usable by several users at the same time or can be restricted so that it can only be accessed if nothing else is using it. Things are identified by their name and have a version number which is returned by the function TH_VER$. The version number of a Thing can be something like 1.03, or it can actually be representative of the functions provided in this version (eg. 1001100) - although it is not certain if this second type of 'version number' will be correctly returned by the current version of TH_VER$, since at the time of writing we have not come across anything which uses this.

If a Thing was not found in memory or another error occurred, TH_VER$ will return the standard error code (see ERNUM).

**Example**

The Hotkey System (HOT_REXT), a part of the Extended Pointer Environment (regarded as standard today), is installed as a Thing. Get its version with:

```
PRINT TH_VER$ ("HOTKEY")
```

**NOTE 1**

In versions prior to version 1.02, this function could return the wrong value for some Things.

**NOTE 2**

The current version of this command will not work on a THOR XVI computer.

**CROSS-REFERENCE**

*THING*, *TH_FIX*.

## 27.10 TINY_EXT

| | |
|---|---|
| Syntax | TINY_EXT |
| Location | TinyToolkit |

This command installs/updates the extensions provided by the Tiny Toolkit. TinyToolkit and Toolkit II have some commands in common (eg. REPORT). If you prefer to use Toolkit II's REPORT command you will generally need to install TK2_EXT after TINY_EXT (on post JM ROMs the Toolkit which was installed second will have priority!). Prior to JS ROMs, the first version of a command loaded as a toolkit has priority.

**NOTE**

Updating TinyToolkit is different from updating other Toolkits with _EXT type commands, in that TinyToolkit simply adds its commands' names to the name list and does not check to see if they were already present. SXTRAS and EXTRAS will list commands twice (or more) and each time that TINY_EXT is issued, memory will be used up (max. 1 KB). Actually, the Toolkit is only present in one place in memory because duplicated commands are stored at the same place in RAM. This problem can be cured with TINY_RMV.

**CROSS-REFERENCE**

*TK2_EXT* updates Toolkit II, *Beule_EXT* the Beule Toolkit. *TINY_RMV* removes most extensions of TinyToolkit from the name list.

## 27.11 TINY_RMV

| | |
|---|---|
| Syntax | TINY_RMV |
| Location | TinyToolkit |

This command removes most of TinyToolkit's commands.

**NOTE**

You should not really use TINY_RMV because the extensions are not removed from the Name List but overwritten with undefined strings. Depending on the operating system and programming environment it may not be possible to re-activate TinyToolkit and internal system conflicts are possible.

**CROSS-REFERENCE**

Re-activate the Toolkit with *TINY_EXT*.

## 27.12 TK2_EXT

| | |
|---|---|
| Syntax | TK2_EXT |
| Location | Toolkit II |

As with other Toolkits, Toolkit II has to be linked into the computer (except on the ST/QL Emulator and under SMSQ/E where it is automatically linked in when the computer is started). This command forces all of the Toolkit II commands to link themselves into the operating system, overwriting existing definitions of any commands with the same name.

**NOTE**

TK2_EXT contains special code to enable Toolkit II commands to be used on JM (and earlier) ROMs in the same program as the TK2_EXT command.

**CROSS-REFERENCE**

See *TINY_EXT*.

## 27.13 TK_VER$

| | |
|---|---|
| Syntax | TK_VER$ |
| Location | Turbo Toolkit |

This function returns the version ID of the Turbo Toolkit, eg. 3e27

**NOTE**

Before v3.00 the Turbo Toolkit did not install properly under Minerva and SMS.

## 27.14 TNC

| | |
|---|---|
| Syntax | TNC (i,n) |
| Location | Toolfin |

The function TNC returns the value of: $n*((1+i)^{1/n}-1)$ where i and n can be any floating point numbers (see MT for error handling).

**CROSS-REFERENCE**

*MT*, *VA*, *VFR*, *VAR*, *TCA*, *TEE*, *RAE*, *RAFE*

## 27.15 TO

| | |
|---|---|
| Syntax | … TO line (GO TO) or<br>TO column (Separator) |
| Location | QL ROM |

This keyword has two uses. The first syntax operates as part of the keyword GO TO. The second syntax is used as a separator in the commands PRINT and INPUT (and also in some toolkit extensions). We shall only deal with the use of TO for PRINT and INPUT here.

As a separator, TO can be very useful for placing data into columns. Its effect is to place the text cursor onto the specified column, or if the text cursor is already at or past that column, then the text cursor is moved one column to the right. This means for instance that:

```
PRINT TO 0
```

will always leave the leftmost column blank!

TO is also affected by the WIDTH setting on non-screen devices. If the specified column is greater than the WIDTH value, the text cursor will be placed onto the next line. On screen devices, if the specified column is too great to fit in the window, the text cursor is placed onto the next line rather than causing an error - note however, that TO carries on counting!!.

TO has no meaning on its own and will cause the error 'Bad Line' if entered on its own.

**NOTE**

On the THOR XVI, if the cursor is already at or past the given column, the text cursor is not moved, in contrast to all other implementations. Programs compiled with Turbo will however display the text as per the standard QL implementation.

**CROSS-REFERENCE**

See *GO TO* and *PRINT*, *INPUT*. *AT* and *CURSOR* allow you to position the text cursor more precisely.

## 27.16 TOP_WINDOW

| | |
|---|---|
| Syntax | TOP_WINDOW [#ch] |
| Location | all THORs |

This command is similar to the PICK command provided by Qjump's QPTR package on the QL. This command brings the specified window (default #1) to the top of the display pile. Under the THOR's windowing system (when this is enabled), as with the Pointer Environment, a program cannot access a window which is partly or fully hidden from view. This command allows the program to force the given window to the top of the pile, thus allowing it to be seen on screen and therefore open to access. If possible, the keyboard queue is also connected to the window, so it is as if the Job has been 'picked' using the keys CTRL C.

**CROSS-REFERENCE**

*WINDOW* allows you to re-position a window. *PIE_ON* allows programs to continue even though their windows are buried under the Pointer Interface. *PICK%* is similar. *POKE* SYS_VARS+133 allows you to enable / disable the THOR's windowing system.

## 27.17 TPFree

| | |
|---|---|
| Syntax | TPFree |
| Location | BTool |

The function TPFree returns a slightly larger or equal value than FREE and FREE_MEM. The reported free memory is available for new jobs.

## 27.18 TRA

| | |
|---|---|
| Syntax | TRA table1 [,table2] or<br>TRA [table1] ,table2 |
| Location | QL ROM (post JM Version) |

This command allows you to perform various translations on data that is passing through the serial ports. It is however one of the most difficult commands in SuperBASIC to use.

The use of TRA will (on non-SMS implementations) affect all data which is sent through the serial ports after the command has been issued, translating bytes whether they are screen dumps, printer control codes, or letters of the alphabet.

The two parameters are addresses of two tables, table1 which contains details of translations to be carried out on both incoming and outgoing data and table2 which contains details of various messages used by the system. Both tables are recognised by the word 19195 ($4AFB) at their start. If either parameter is not specified, then the default value of -1 is assumed, which tells QDOS to leave that translation table alone.

When QDOS is first initiated translation is not enabled, which means that data passing through the serial ports is unaffected. You can revert to this situation by using the command TRA 0. You can also revert to the original error messages with TRA ,1 (use TRA 0,1 to reset both to their original status). The English character set is used in all ROM implementations of the QL (no matter which country the machine is set up for). However, you can select to use the 'local' character set for serial communication purposes if you wish by using the command TRA 1 which tells QDOS to use the 'local' translation table (this has no effect on UK ROMs).

The two translation tables have different formats and uses, depending on whether the serial ports are being used for transmission or receipt of data. We therefore deal with each table separately. Note that TRA is implemented differently on THORs and SMS see the separate notes on the make up of their translation tables.

**Table 1**

Table 1 is actually split into two lists:

- Transa contains a list of single character conversions;
- Transb which contains a list of multiple character conversions.

As to which list is used depends on whether the channel is sending or receiving data:

1. If the channel is sending data, the outgoing character is first translated according to Transa, using the character code as an index. If the resulting value is a zero, Transb is scanned for the proper entry. However, if the resulting value is non-zero, then this is used as a replacement for the byte to be sent.
2. When receiving, only Transa is used. The table is scanned cyclically starting at the received character's position until a position is found containing the received value. The translated value will be this position index. If the received value is not found in the table, the value itself is used.

The physical format of table1 is as follows:

```
Table1 Word      19195
       Word      Offset of Transa from Table1 (Transa-Table1)
       Word      Offset of Transb from Table1 (Transb-Table1)

Transa 256 bytes (see below)
Transb Byte      Number of multiple translations or 0 x bytes(see below)
```

Transa is a 256 byte list of character substitute codes for each character code from 0 to 255. If you wish to use multiple translates for a given character, then you will need to insert 0 in the appropriate place in this list.

Transb is a table of multiple translations (which can only be used in transmit mode). It is made up of four bytes for each translate, being the code to be translated, followed by three replacement codes. If you do not need three replacement codes, the unused ones should be zero. Unfortunately, you cannot combine the effects of these various translations (see the second example below).

**Table 2**

Table 2 allows you to set the various system and error messages used by QDOS (for example to implement other languages). The format of Table2 is even more complex:

```
Table2  Word      19195
        Word      Offset of error1 from Table2 (error1-Table2)
        Word      Offset of error2 from Table2 (error2-Table2)
        ....
        Word      Offset of error20 from Table2 (error20-Table2)
        Word      Offset of error21 from Table2 (error21-Table2)
        Word      Offset of mess1 from Table2 (mess1-Table2)
        Word      Offset of mess2 from Table2 (mess2-Table2)
        ....
        Word      Offset of mess7 from Table2 (mess7-Table2)
        Word      Offset of mess8 from Table2 (mess8-Table2)

error1  Word      Length of string
        Bytes     String forming message for 'not complete'

error2  Word      Length of string
        Bytes     String forming message for 'invalid job'

        ....

error21 Word      Length of string
        Bytes     String forming message for 'Bad Line'

mess1   Word      Length of string
        Bytes     String to replace 'At line ' (***)

mess2   Word      Length of string
        Bytes     String to replace ' sectors'

mess3   Word      Length of string
        Bytes     String to replace 'F1 .. monitor F2 .. TV ' (***)

mess4   Word      Length of string
        Bytes     String to replace '© 1983 Sinclair Research Ltd' (***)

mess5   Word      Length of string
        Bytes     String to replace 'during WHEN processing'

mess6   Word      Length of string
        Bytes     String to replace 'PROC/FN cleared'

mess7   Bytes     String to replace 'SunMonTueWedThuFriSat' (***)
```

(continues on next page)

(continued from previous page)

```
mess8   Bytes     String to replace 'JanFebMarAprMayJunJulAugSepOctNovDec'␣
↪(***)
```

Please note that all strings *other* than those marked (***) *must* end with a newline, CHR$(10).

Also please also note the differing format of mess7 and mess8.

Although the THOR computers support both of the above table formats, the THOR has extended the usefulness of TRA in order to allow you to send longer strings of characters for each translation. On the other hand, SMS has implemented a different way of amending the messages generated by the operating system (see below). Examples of the standard format follow:

**Example 1**

A program to change all of the error messages to more meaningful messages:

```
100 Chk$=VER$
105 IF Chk$='AH' OR Chk$='JM': PRINT'Not supported'
110 table2=ALCHP(1024)
120 RESTORE
130 POKE_W table2,19195
140 mess_add=table2+30*2
150 FOR errx=1 TO 29
160 POKE_W table2+errx*2,mess_add-table2
170 READ mess$
180 IF errx<28
190 SELect ON errx: =1 TO 21,23,26 TO 27: mess$=mess$&CHR$(10)
200 POKE_W mess_add,LEN(mess$): mess_add=mess_add+2
210 END IF
220 FOR move_mess=1 TO LEN(mess$)
230 POKE mess_add,CODE(mess$(move_mess)): mess_add=mess_add+1
240 END FOR move_mess
250 overf=mess_add/2:IF overf<>INT(overf): mess_add=mess_add+1
260 END FOR errx
270 TRA 0,table2
280 DATA 'Operation Not Complete'
290 DATA 'Job Does Not Exist'
300 DATA 'Insufficient Memory'
310 DATA 'Parameter Outside Permitted Range'
320 DATA 'Buffer Full'
330 DATA 'Channel Not Open'
340 DATA 'File or Device Not Found'
350 DATA 'File Already Exists'
360 DATA 'File or Device In Use'
370 DATA 'End of File'
380 DATA 'Drive Full'
390 DATA 'Invalid File or Device Name'
400 DATA 'Transmit Error'
410 DATA 'Format Failed'
420 DATA 'Invalid Parameter'
430 DATA 'Filing System Medium Check Failed'
```

(continues on next page)

(continued from previous page)

```
440 DATA 'Invalid Expression'
450 DATA 'Maths Overflow'
460 DATA 'Operation Not Implemented'
470 DATA 'Read Only Device'
480 DATA 'Invalid Syntax'
490 DATA 'At line '
500 DATA ' sectors'
510 DATA 'F1 .. monitor'&CHR$(10)&'F2 .. TV'
520 DATA '©1983 Sinclair Research Ltd.'
530 DATA 'During WHEN processing'
540 DATA 'PROC/FN Definition Cleared'
550 DATA 'SunMonTueWedThuFriSat'
560 DATA 'JanFebMarAprMayJunJulAugSepOctNovDec'
```

**Example 2**

A short program to allow you to print pound signs (£) from SuperBASIC (this assumes an Epson compatible printer which is set up in US ASCII mode):

```
100 table1=ALCHP(1024)
110 POKE_W table1,19195
120 Transa=table1+6
130 Transb=Transa+256
140 FOR i=0 TO 255:POKE Transa+i,i
150 POKE_W table1+2,Transa-table1
160 POKE_W table1+4,Transb-table1
170 POKE Transb,3
175 POKE Transa+128,0: POKE Transa+129,0: POKE Transa+CODE('£'),0
180 POKE Transb+1,128
190 POKE Transb+2,27: POKE Transb+3,CODE('R'): POKE Transb+4,3
200 POKE Transb+5,129
210 POKE Transb+6,27: POKE Transb+7,CODE('R'): POKE Transb+8,0
215 POKE Transb+9,CODE('£')
216 POKE Transb+10,128: POKE Transb+11,CODE('#'): POKE Transb+12,129
220 TRA table1,0
```

Unfortunately, despite lines 215 and 216, the command:

```
OPEN #3,ser1: PRINT #3,'£'
```

will still fail to produce a pound sign on your printer (you will get a single quote mark normally).

This demonstrates the fact that you cannot link translates. To get a pound sign, you will need to use the line:

```
OPEN #3,ser1: PRINT#3,CHR$(128) & '£' & CHR$(129)
```

Indeed, because of the nature of the translation tables, the following has exactly the same effect as the above program:

```
100 table1=ALCHP(1024)
110 POKE_W table1,19195
120 Transa=table1+6
130 Transb=Transa+256
140 FOR i=0 TO 255:POKE Transa+i,i
150 POKE_W table1+2,Transa-table1
160 POKE_W table1+4,Transb-table1
170 POKE Transb,2
180 POKE Transa+128,0:POKE Transa+129,0
190 POKE Transa+CODE('£'),CODE('#')
200 POKE Transb+1,128
210 POKE Transb+2,27: POKE Transb+3,CODE('R'):POKE Transb+4,3
220 POKE Transb+5,129
230 POKE Transb+6,27: POKE Transb+7,CODE('R'):POKE Transb+8,0
240 TRA table1,0
```

**NOTE 1**

An extended serial driver is available in the public domain which enables Minerva machines and Amiga QDOS to use a translation table the same as the extended translation table provided on the THOR XVI.

**NOTE 2**

On Minerva ROMs (v1.83 or earlier), there are problems when using TRA with only one parameter.

**NOTE 3**

JS ROMs have problems in translating characters above CHR$(127)

**SMS NOTES**

SMS supports the standard format table1. However, the messages cannot be altered using table2 - use LANG_USE for this. As with the original version, if table1 is specified to be 0, this will deactivate the translation. However, it does not smash the pointer to a user-defined translation routine which can then be re-activated with TRA 1 (compare the original version where you would need to re-run the program setting up the user-defined translation table).

SMS also allows you to have language dependent translation tables (linked to one of the languages currently loaded - see LANG_USE). To enable these, use the command:

```
TRA 1,lang
```

where lang is the Car Registration Code or Language code of the country.

```
TRA 0,lang
```

will set up the relevant translation table, ready to be enabled with TRA 1.

There are also several in-built language independent translate tables which are accessed by setting table1 to small values. The dip-switches on your printer need to be set to USA. Currently there are only two language independent translate tables supported (so far as we are aware):

- The command TRA 3 will enable IBM Graphics translation table:
  - QDOS CHR$(HEX('C0')) to CHR$(HEX('DF')) and CHR$(HEX('F0')) to CHR$(HEX('FF')) are passed through the channel unchanged.

  - CHR$(HEX('E0')) to CHR$(HEX('EF')) are translated to represent CHR$(HEX('B0')) to CHR$(HEX('BF')) respectively.
  - As from v2.50, the paragraph sign, CHR$(HEX('15')) is also passed through unaffected.
- The command TRA 5 will enable GEM VDI translation table:
  - Here QDOS CHR$(HEX('C0')) to CHR$(HEX('FF')) are passed through the port unchanged.

Also please note that under SMS, TRA will only affect channels which are OPENed after the TRA command, or channels which have already been OPENed with TRA active. In any case, TRA 0 never affects OPEN channels. TRA address will also not affect OPEN channels which have been affected by TRA 0. Note however that changing the BAUD rate will affect the translate on ALL channels.

**SMS Example**

```
TRA 1: REMark Enable translate table for Country set up by default.
TRA 1,F: REMark Enable French Translation table.
TRA 0: REMark Disable Translate Tables.
TRA 1: REMark Re-enable French Translation Table
```

**THOR XVI NOTES**

The THOR XVI supports both the standard translation format above and also an expanded Translation Table, which replaces Table1 by a larger table in the following format:

**Thor Table1**

The format of the new expanded Translation Table is:

```
Table1  Longword  $4AFB0001 Distinguishes the new table from the old one.
        Word      Offset of Transa from table1 (Transa-table1)
        Word      Offset of Transb from table1 (Transb-table1)
        Longword  Offset of Pream from table1 (Pream-table1)
        Longword  Offset of Post from table1 (Post-table1)

Transa  256 Bytes (See below)
Transbx Bytes     (See below)
Pream   Word      Length of preamble string
        Bytes     String to be sent when channel is opened
Post    Word      Length of postamble string
        Bytes     String to be sent when channel is closed
```

The format of Transa and Transb is slightly different from the standard translation table:

Transa is a 256 byte list of one character conversions, with an entry of zero if Transb is to be used.

Transb is however much more complex as each entry is made up of the following (allowing a string of up to 255 characters to be sent as a replacement for the given character):

```
Transb  Byte      Character to be replaced
        Byte      Length of a string to replace character x
        Bytes     A string (up to 255 characters long) to replace the given␣
↪character.
```

The last entry in this list must be 0,1,0 to allow nul characters to be sent.

Transb is generally therefore in the following format:

```
Transb   x Bytes ch1,len1,'text1'
         x Bytes ch2,len2,'text2'
         ....
         x Bytes chn,lenn,'textn'
         x Bytes 0,1,0
```

**THOR Example**

For example, following upon our earlier example, one entry in Transb would allow for trouble-free translation of the pound sign. This could therefore be achieved by the program listed below:

```
100 table1=ALCHP(1024)
110 POKE_L table1,HEX('4AFB0001')
120 Transa=table1+16
130 Transb=Transa+256
140 FOR i=0 TO 255: POKE Transa+i,i
150 POKE_W table1+4,Transa-table1
160 POKE_W table1+6,Transb-table1
170 POKE_L table1+8,0
180 POKE_L table1+12,0
190 POKE Transa+CODE('£'),0
200 POKE Transb,CODE('£')
210 POKE Transb+1,7
220 POKE Transb+2,27: POKE Transb+3,CODE('R'): POKE Transb+4,3
230 POKE Transb+5,CODE('#')
240 POKE Transb+6,27: POKE Transb+7,CODE('R'): POKE Transb+8,0
250 POKE Transb+9,0: POKE Transb+10,1: POKE Transb+11,0
260 TRA table1,0
```

The preamble and postamble entries allow you to set up the printer when the channel is opened or closed. These can both be up to 32767 characters long.

From version 6.41, the TRA command has been enhanced to make extra use of the various different character sets supplied as standard on this QDOS implementation. The Russian, Russisk and Greek language set-ups now use a table converting $80 … $BF to $60 … $DF to allow use with down-loaded character sets or Brother/HP Laser Jet + laser printers, where codes $80 … $9F are often treated as control codes.

The default translate table (TRA 1) now works reasonably with ISO codes, allowing printers to be set in the appropriate language range. This works okay with the French, Danish, Spanish, Japanese, and German set-ups (except for the paragraph character in German). On the Swedish language set-up, only U/u umlaut (Ü/ü) does not work and the Italian language set-up fails on e grave (é), u and a acute (ú and á), due to the conflict with French.

A special extended translation table will always be required for the Russisk, Russian and Greek language set-ups, depending on the type of printer connected to the system.

**CROSS-REFERENCE**

Please refer to the Appendix concerning serial and parallel device drivers.

## 27.19 TRIM$

| | |
|---|---|
| Syntax | TRIM$ (string$) |
| Location | TRIM |

The function strips off all preceding and appended spaces from a string and returns the result of this. Any string can be used as a parameter.

**Examples**

```
TRIM$(" Hello World"): REMark = "Hello World"
TRIM$("second try "): REMark = "second try"
TRIM$(" "): REMark = ""
TRIM$(""): REMark = ""
TRIM$(CHR$(27)): REMark = CHR$(27)
```

**CROSS-REFERENCE**

*LEN* returns the length of a string.

## 27.20 TRINT

| | |
|---|---|
| Syntax | TRINT (x) |
| Location | TRIPRODRO |

The function TRINT gives the integer part of a floating point number, it differs from INT for negative numbers only: INT always returns the next lowest integer, this is the same as the integer part for positive numbers; however below zero INT always returns one less than TRINT. For example:

```
INT(-PI)
```

will return -4 and:

```
TRINT(-PI)
```

will return -3.

**CROSS-REFERENCE**

The fact that:

```
x = TRINT(x) + FRACT(x)
```

can be exploited to substitute one of the two functions by the other, for example:

```
100 DEFine FuNction MYTRINT(x)
110   RETurn x - FRACT(x)
120 END DEFine MYTRINT
```

If you want to round numbers, refer to *DROUND* and *PROUND*.

## 27.21 TROFF

| | |
|---|---|
| Syntax | TROFF |
| Location | Minerva (TRACE) |

This command turns off the trace function and closes any file associated with the trace output.

**CROSS-REFERENCE**

*TRON* and *SSTEP* turn the trace function on.

## 27.22 TRON

| | |
|---|---|
| Syntax | TRON [ {#ch \| device_file}] [; [first] [TO [last]]] |
| Location | Minerva (TRACE) |

This command is very similar to SSTEP except that it does not wait for a key to be pressed before each statement is executed.

**NOTE**

Minerva's TRACE Toolkit is quite useful but is still just a simple demonstration of an extension which has been internally added to the SuperBASIC code.

**CROSS-REFERENCE**

See *TROFF* and *SSTEP*.

## 27.23 TRUE%

| | |
|---|---|
| Syntax | TRUE% |
| Location | TRUFA |

TRUE% is the constant 1. It is used to write programs which are more legible or which adopt habits from the PASCAL language.

**Example**

```
IF QuATARI=TRUE% THEN ...
```

is the same as:

```
IF QuATARI THEN ...
```

**CROSS-REFERENCE**

*FALSE%* is 0. *SET* can be used to create constants as resident keywords.

## 27.24 TRUNCATE

| | |
|---|---|
| Syntax | TRUNCATE #channel [\position] or TRUNCATE |
| Location | Toolkit II, THOR XVI |

Every file has a certain length, measured in bytes, which can be reduced with the command TRUNCATE. If TRUNCATE is used without the position parameter, the end of the file will be moved to the current file pointer position, meaning that for most purposes, the last byte of the file is the byte which was being pointed to.

If you supply a second parameter, then the file pointer is set to the given position before the file is TRUNCATEd. Note that any data after the new 'end of file' will be lost.

TRUNCATE returns error -15 (invalid parameter) if the specified channel is not actually linked to a file. A position greater than the actual file length, such as position>=FLEN(#channel) has no effect. TRUNCATE without any parameters uses #3 as the default channel and is therefore the same as:

```
TRUNCATE #3
```

**NOTE**

The syntax TRUNCATE \position is not valid, error -17 (error in expression) will be reported. You have to specify a channel number if you intend to set the file pointer before truncating the file.

**CROSS-REFERENCE**

*FLEN* and *FILE_LEN* return the length of a file, *FPOS* and *FILE_POS* the current file pointer position, *FILE_PTRA* and *FILE_PTRR* move the file pointer as do *GET*, *PUT*, *BGET* and *BPUT*.

## 27.25 TTALL

| | |
|---|---|
| Syntax | TTALL (space [,jobid] ) |
| Location | QView Tiny Toolkit |

This function is the same as ALCHP but memory allocated with TTALL cannot be cleared with CLCHP or RECHP: TTREL must be used on the return value of TTALL; see TTFINDM for an example.

**CROSS-REFERENCE**

*TTREL* See also *RESERVE*.

## 27.26 TTEDELETE

| | |
|---|---|
| Syntax | TTEDELETE (file$) |
| Location | QView Tiny Toolkit |

This is a function analogous to the command DELETE - it will return the QDOS error code. The default device is not supported, ie. the file name must be specified absolutely.

**NOTE**

In contrast to DELETE, TTEDELETE will return the value -7 if the file did not exist.

**CROSS-REFERENCE**

*DELETE* of course.

## 27.27 TTEFP

| | |
|---|---|
| Syntax | TTEFP (floatvar, floatstr$) |
| Location | QView Tiny Toolkit |

This function tries to convert the string given as the second parameter into a floating point number and assign this value to the floating point variable given as the first argument. There is no difference to the assignment:

```
floatvar = floatstr$
```

except where an error occurs, ie. if floatstr$ cannot be converted to a float. Whereas the assignment above will break with an error, TTEFP will allow you to track that down by checking its return; the number returned by TTEFP is the QDOS error code (or 0 if the assignment was successful).

**Example**

A piece of code which asks for the age of the user would look similar to this:

```
100 CLS
110 REPeat question
120   INPUT "How old are you?"!age$
130   ec = TTEFP(age, age$)
140   SELect ON ec
150     = 0: IF age < 13 OR age > 100 THEN
160            PRINT "You're surely kidding!!"
170          ELSE EXIT question
180          END IF
190     = -17: PRINT "Digits, not letters, ok?"
200     = -18: PRINT "Reasonable numbers, please."
210     = REMAINDER : PRINT "What's this about?"
220   END SELect
230 END REPeat question
240 PRINT "So you are"!age!"years old... :-)"
```

**CROSS-REFERENCE**

*CHECK%*, *CHECKF*.

## 27.28 TTEOPEN

| | |
|---|---|
| Syntax | TTEOPEN (#channel [,openmode], device$) |
| Location | QView Tiny Toolkit |

The TTEOPEN function opens the specified #channel to any device given as a string. The type of open is optional and ranges from 0 to 4 - the meaning is the same as for Minerva's extended OPEN or FILE_OPEN. If TTEOPEN is called from the interpreter (Multiple BASICs included) then channel must either be an existing channel number (which would be then closed by TTEOPEN prior to being reopened) or lower than the highest channel number currently used: TTEOPEN will break with 'bad parameter' if that is not the case.

**CROSS-REFERENCE**

*OPEN*, *FILE_OPEN* and the various FOP_XXX keywords.

## 27.29 TTET3

| | |
|---|---|
| Syntax | TTET3 ( [#ch,] [timeout%,] trapno%, bufadr) |
| Location | QView Tiny Toolkit |

This is a really extraordinary function because it allows you to call the TRAP #3 operating system calls which handle screen devices, so you would not theoretically need many other commands other than this one to manipulate windows, if the use of TTET3 were not complicated by the nature of its design.

The function TTET3 should only be used by experienced users (except for some fool-proof usages shown in the examples), so do not worry if you do not understand the following… although we have tried to keep it simple.

Let's first turn to the syntax:

- The channel #ch (default #1) must refer to a window (con_ or scr_).
- The timeout for the machine code call trap is optional, the default is -1 (that means the operating system will try indefinitely to execute the trap) which is fine for most purposes.
- Trapno% is a small positive integer that identifies the trap.
- Bufadr must point to a piece of memory at least 16 bytes long.

Since this toolkit provides its own buffer starting at TTV, it is recommended and safe to use this for bufadr.

The required 16 bytes buffer is used to communicate with the processor, the registers D1, D2, A1 and A2 occupy four bytes (one longword) each within the buffer - they are copied to the processor when the trap is executed and after the trap has finished will hold any return values and be copied back into the buffer so that they may be read with the lines:

```
D1=PEEK_L(bufadr)
D2=PEEK_L(bufadr+4)
A1=PEEK_L(bufadr+8)
A2=PEEK_L(bufadr+12)
```

**Example 1**

Superfluous with CLS but:

```
x=TTET3(#2,32,TTV)
```

does a:

```
CLS#2.
```

**Example 2**

The procedure SD_ENQUIRE reads the window size and cursor position, the values are placed in the passed integer variables. You can test if anything went wrong (eg. #ch does not refer to a window) by checking if any of the values returned are negative.

The parameter what% determines the units,

- what% = 0 will have the effect that wsx% and wsy% are the window width and height in pixels and that (cpx%, cpy%) is the position of the text cursor in screen pixels;
- what%<>0 will give the same information but in characters.

```
100 FOR i = 0, 1
110   SD_ENQUIRE #2, i, a%, b%, c%, d%
120   PRINT a%, b%, c%, d%
130 END FOR i
140 :
150 DEFine PROCedure SD_ENQUIRE (ch, what%, wsx%, wsy%, cpx%, cpy%)
160   LOCal trapno%
170   POKE_L TTV+8, TTV+16
180   trapno% = 10 + NOT(NOT what%)
190   IF TTET3(#ch, 100, trapno%, TTV) THEN
200     wsx% = -1: wsy% = -1: cpx% = -1: cpy% = -1
210     RETurn
220   END IF
230   wsx% = PEEK_W(TTV+16): wsy% = PEEK_W(TTV+18)
240   cpx% = PEEK_W(TTV+20): cpy% = PEEK_W(TTV+22)
250 END DEFine SD_ENQUIRE
```

On Minerva, you can write NOT (NOT what%) without brackets. SD_ENQUIRE is absolutely clean, there is no danger at all that the system might crash, that it does not run on all QDOS machines or anything like that.

All other machine code traps available through TTET3 are covered by commands in this manual, but TTET3 can be used to avoid the need to link in a Toolkit.

**CROSS-REFERENCE**

Please refer to system documentation for details on each trap! See also *IO_TRAP*, *QTRAP* and *MTRAP*.

## 27.30 TTEX

| | |
|---|---|
| Syntax | TTEX file$ [;cmd$] |
| Location | QView Tiny Toolkit |

This command is analogous to EXEC - like EX, a command string can be passed to the program. However, unlike EX, default devices, pipes and channel passing are not supported.

**CROSS-REFERENCE**

See *TTEX_W* and *EX*.

## 27.31 TTEX_W

| | |
|---|---|
| Syntax | TTEX_W file$ [;cmd$] |
| Location | QView Tiny Toolkit |

This bears the same relation to EXEC_W and EW as TTEX does to EXEC and EX.

**CROSS-REFERENCE**

See *TTEX* and *EW*.

## 27.32 TTFINDM

| | |
|---|---|
| Syntax | TTFINDM (addr, length, tosearch$) |
| Location | QView Tiny Toolkit |

This function will search for a given string in memory, see SEARCH, MSEARCH and BLOOK. Memory is scanned from address addr for length bytes onwards. The search is case-sensitive. TTFINDM returns zero if the string was not found or the positive relative address plus one where the string first occurs.

**Example**

Old or badly written programs and Toolkits require the screen located at address $20000 and the System Variables at $28000, this causes great problems an Minerva in Dual Screen Mode and other advanced systems as well.

Our demonstration for TTFINDM loads a file into memory and scans it for the occurrence of the two mentioned numbers in their internal format. This method of checking code is pretty reliable for hand-written machine code. The problem$ values have been computed with:

```
MKL$(HEX("20000"))
```

and

```
MKL$(HEX("28000"))
```

```
100 file$ = "flp2_tool_shape_cde"
110 length = FLEN(\file$)
120 DIM problem$(2,4)
130 problem$(1) = CHR$(0)&CHR$(2)&CHR$(0)&CHR$(0)
140 problem$(2) = CHR$(0)&CHR$(2)&CHR$(128)&CHR$(0)
150 :
160 PAPER 3: CLS: INK 7
170 PRINT "Allocating memory...";
180 adr = TTALL(length): PRINT "done"
190 IF adr = 0 THEN PRINT "No memory.": STOP
200 PRINT "Loading"!file$;"...";
210 LBYTES file$ TO adr: PRINT "done"
220 FOR test = 1 TO DIMN(problem$)
230   PRINT "Test"!test;"...";
```

(continues on next page)

(continued from previous page)

```
240   found = TTFINDM(adr, length, problem$(test))
250   IF found THEN
260     PRINT "failed"
270     DUMPIT adr+found-1, 4, 20
280   ELSE PRINT "ok"
290   END IF
300 END FOR test
310 PRINT "Releasing memory...";
320 TTREL adr: PRINT "done"
330 :
340 DEFine PROCedure DUMPIT (adr, length%, surr%)
350   INK 4: PRINT TTPEEK$(adr-surr%, surr%);
360   INK 7: PRINT TTPEEK$(adr, length%);
370   INK 4: PRINT TTPEEK$(adr+length%, surr%): INK 7
380 END DEFine DUMPIT
```

**CROSS-REFERENCE**

*SEARCH*, *BLOOK*, *MSEARCH* are all similar.

## 27.33 TTINC

| | |
|---|---|
| Syntax | TTINC #ch, xsp%, ysp% |
| Location | QView Tiny Toolkit |

This command is identical to CHAR_INC.

## 27.34 TTME%

| | |
|---|---|
| Syntax | TTME% |
| Location | QView Tiny Toolkit |

This function gives the job number of the current job.

**CROSS-REFERENCE**

See *JOBS* for information about *TTME%*'s return.

## 27.35 TTMODE%

| | |
|---|---|
| Syntax | TTMODE% |
| Location | QView Tiny Toolkit |

This is the same as RMODE.

## 27.36 TTPEEK$

| | |
|---|---|
| Syntax | TTPEEK$ (adr, length) |
| Location | QView Tiny Toolkit |

See PEEK$.

**Example**

```
PRINT TTPEEK$(TTV-2,2)
```

always shows the letters QV.

## 27.37 TTPOKEM

| | |
|---|---|
| Syntax | TTPOKEM adr2 { , \| ! \| TO } adr1, bytes |
| Location | QView Tiny Toolkit |

The command TTPOKEM moves any amount of bytes in memory from address adr1 to adr2. The choice of the separator only makes a difference if the source memory area overlaps with the destination. The separator has the following effects:

- Comma (,) : the move is non-destructive, meaning that the memory area from adr1 has been copied to adr2 so that it is identical to the area which was previously located at adr1 (the area at adr1 has changed of course if the areas overlap).
- ! or TO : The move is destructive and the overlapping parts of or both blocks will be messed up, that is because the first few bytes stored at adr1 will be stored at adr2 onwards, thus overwriting the last few bytes of adr1 which should have been copied.

**CROSS-REFERENCE**

*BMOVE*, *COPY_B*, *COPY_L*, *COPY_W*

## 27.38 TTPOKE$

| | |
|---|---|
| Syntax | TTPOKE$ adr, string$ |
| Location | QView Tiny Toolkit |

This is the same as POKE$.

## 27.39 TTREL

| | |
|---|---|
| Syntax | TTREL adr |
| Location | QView Tiny Toolkit |

This is similar to the RECHP command, except that it will only remove areas set aside with TTALL.

**CROSS-REFERENCE**

*TTALL*. See also *DISCARD*.

## 27.40 TTRENAME

| | |
|---|---|
| Syntax | TTRENAME file1$, file2$ |
| Location | QView Tiny Toolkit |

This command is similar to RENAME except that no default devices are supported. Toolkit II (which apart from providing the SuperBASIC keyword RENAME adds an operating system extension to rename files) is not required.

## 27.41 TTSUS

| | |
|---|---|
| Syntax | TTSUS frames |
| Location | QView Tiny Toolkit |

The command TTSUS will cause the current job to be suspended for frames/50 seconds (frames/60 on some QLs), ie. the job will wait at the TTSUS command for the specified time and then continue with the next command. It is suggested that TTSUS is used as an alternative to the PAUSE command (same parameter) because it does not require an open channel - it's a good idea, but please take into account that pressing a key will not break the pause generated by TTSUS.

**CROSS-REFERENCE**

*SJOB*, *PRIO*, *PAUSE*

## 27.42 TTV

| | |
|---|---|
| Syntax | TTV [ ( x1 $^{*}$[,$x^{i}$]$^{*}$ )] |
| Location | QView Tiny Toolkit |

The function TTV returns the address of the QView Toolkit workspace, which is a piece of shared memory of 176 bytes which can be accessed from any job. The idea is that this workspace is used for communication between different parts of the same program. By default, these bytes are set to zero, so that you can freely POKE to them without the danger of crashes. Note that the value of TTV is the same for all jobs. The parameters are (more or less) just for fun, their sum is added to the start address of the QView Toolkit workspace before that address is returned. So:

```
TTV = TTV(0)
TTV(10) = TTV+10 = TTV(3,3,3,1)
```

**Example**

The workspace is preceded by 64 bytes for QView Toolkit's internal use. There is however one value that is interesting to look at:

```
PEEK_L(TTV-64)
```

is a very precise counter, it increases once every frame. This is ideal for checking program speed without the need for long lasting benchmarks, the following programs demonstrates the difference in speed between some different types of FOR constructions:

```
100 TIMER_START
110 FOR i = 1 TO 10000
120   REMark
130 END FOR i
140 TIMER_STOP
150 :
160 TIMER_START
170 FOR i = 1 TO 10000: REMark
180 TIMER_STOP
190 :
200 TIMER_START
210 FOR i% = 1 TO 10000: REMark
220 TIMER_STOP
230 :
240 :
250 DEFine PROCedure TIMER_START
260   POKE_L TTV(-64),0
270 END DEFine TIMER_START
280 :
290 DEFine PROCedure TIMER_STOP
300   LOCal count
310   count = PEEK_L(TTV-64)
320   PRINT INT (count/5) /10;"s"
330 END DEFine TIMER_STOP
```

The third test (lines 200 to 220) works on Minerva and SMS only, and is the fastest: 78% faster than the first test! Some QLs (mainly those in the UK using TV's) will need to amend line 320 to read:

```
320 PRINT INT (count/6 )/10;'s'
```

**CROSS-REFERENCE**

See *T_ON*, *T_OFF*, *T_START* and *T_STOP*

## 27.43 TT$

| | |
|---|---|
| Syntax | TT$ |
| Location | QView Tiny Toolkit |

This function returns the version ID of the QView Tiny Toolkit, eg. QVTK1.3

## 27.44 TURBO_diags

| | |
|---|---|
| Syntax | TURBO_diags " [ d \| i \| o ] " |
| Location | Turbo Toolkit v3.00+ |

This is a directive for the TURBO compiler and should be located at the start of your program before any active program lines. A program can be compiled with line numbers included, which increases the amount of memory and dataspace required by a program, but does mean that if an error occurs, the line number will be displayed. If you do not include line numbers, any errors will report 'at line 0' and ERLIN% will return 0. This directive accepts a single character string which should be one of the following values:

- d: Display line numbers during compilation process but do not include them in final code.
- i: Include line numbers in final code.
- o: Omit line numbers all together.

As with other compiler directives, this value can be changed by configuring the parser_task program or by entering a different value on the Parser's front panel.

**Example**

```
5 TURBO_diags "i"
```

**CROSS-REFERENCE**

See *TURBO_F*,*TURBO_locstr*, *TURBO_model*, *TURBO_objdat*, *TURBO_objdat*,*TURBO_objfil*, *TURBO_optim*,*TURBO_repfil*, *TURBO_struct*, *TURBO_taskn* and *TURBO_window* for other directives

## 27.45 TURBO_F

| | |
|---|---|
| Syntax | TURBO_F |
| Location | Turbo Toolkit v3.00+ |

This directive forms part of the EXTERNAL and GLOBAL Turbo directives and is used to specify the names of FuNctions contained in another compiled module for a program where that program is loaded as several linked modules (using LINK_LOAD) rather than one huge program.

**NOTE**

Before v3.00, this directive was called FUNCTION which caused problems with installing Turbo Toolkit under Minerva and SMS.

**CROSS-REFERENCE**

See *TURBO_locstr* and *TURBO_P* for other directives Refer to EXTERNAL for more information. Use *TK_VER$* to check on the version of TURBO toolkit.

## 27.46 TURBO_locstr

| | |
|---|---|
| Syntax | TURBO_locstr " [ i \| r \| c ] " |
| Location | Turbo Toolkit v3.00+ |

This is a directive for the TURBO compiler and should be located at the start of your program before any active program lines. All strings used within a compiled program should be dimensioned so that the compiler knows the maximum amount of memory which needs to be set aside to store a string. Any attempt to assign a longer value to the string than that set with a DIM or LOCal command will be cut to the appropriate length.

If TURBO has to automatically DIMension a string, it assumes a length of 100 characters (unless configured otherwise).

The TURBO_locstr directive relates to the way in which TURBO should deal with LOCal strings or string parameters. It accepts a single character string which should be one of the following values:

- i: Ignore any strings which are used in the program but not dimensioned. TURBO assumes that you know what you are doing with them.
- r: Report any undimensioned strings - do nothing with them.
- c: Create a DIM statement for any undimensioned strings, making them global sizes for the whole program.

As with other compiler directives, this value can be changed by configuring the parser_task program or by entering a different value on the Parser's front panel.

**Example**

```
5 TURBO_locstr "c"
```

**CROSS-REFERENCE**

See *TURBO_diags*, *TURBO_model*, *TURBO_objdat*,*TURBO_objfil*, *TURBO_optim*, *TURBO_repfil*, *TURBO_struct*, *TURBO_taskn* and *TURBO_window* for other directives

## 27.47 TURBO_model

| | |
|---|---|
| Syntax | TURBO_model " [ < \| > ] " |
| Location | Turbo Toolkit v3.00+ |

This is a directive for the TURBO compiler and should be located at the start of your program before any active program lines. The TURBO compiler is able to generate code using either 16 bit addressing or 32 bit addressing. The former produces more compact and slightly faster code than the latter, but runs into problems if the compiled version of your program (excluding dataspace) is larger than 64K. You should therefore experiment with this setting - if your program is too large to be compiled with 16 bit addressing, the TURBO compiler will report an error during the code generation stage to the effect that the program is 'too large for optimisation'. This does not overcome the problem with running TURBO compiled programs on systems which have a lot of memory or which do not have the system variables stored at $28000. To cover these programs, it is necessary to run them through the TurboPatch program supplied with later versions of the TURBO toolkit.

The TURBO_model directive accepts a single character string which should be one of the following values:

- <: Generate code using 16-bit addressing (shown as <64K on screen).
- >: Generate code using 32-bit addressing.

As with other compiler directives, this value can be changed by configuring the parser_task program or by entering a different value on the Parser's front panel.

**Example**

```
5 TURBO_model "<"
```

**CROSS-REFERENCE**

See *TURBO_diags*, *TURBO_locstr*, *TURBO_objdat*, *TURBO_objfil*, *TURBO_optim*, *TURBO_repfil*, *TURBO_struct*, *TURBO_taskn* and *TURBO_window* for other directives

## 27.48 TURBO_objdat

| | |
|---|---|
| Syntax | TURBO_objdat sizesize=0…850 |
| Location | Turbo Toolkit v3.00+ |

This directive is exactly the same as DATA_AREA.

**NOTE**

This setting will override a previous DATA_AREA directive in the same program. It will also be overridden by a later DATA_AREA directive in the same program.

**CROSS-REFERENCE**

See *TURBO_diags*, *TURBO_locstr*, *TURBO_model*, *TURBO_objfil*, *TURBO_optim*, *TURBO_repfil*, *TURBO_struct*, *TURBO_taskn* and *TURBO_window* for other directives

## 27.49 TURBO_objfil

| | |
|---|---|
| Syntax | TURBO_objfil filename$ |
| Location | Turbo Toolkit v3.00+ |

This is a directive for the TURBO compiler and should be located at the start of your program before any active program lines. This directive expects you to specify a string which will form the filename of the compiled program produced by TURBO. The full filename (including device) should be specified in quote marks. As with other compiler directives, this value can be changed by configuring the parser_task program or by entering a different value on the Parser's front panel.

**Example**

```
5 TURBO_objfil "ram1_CT_exe"
```

**CROSS-REFERENCE**

See *TURBO_diags*, *TURBO_locstr*, *TURBO_model*, *TURBO_objdat*, *TURBO_optim*, *TURBO_repfil*, *TURBO_struct*, *TURBO_taskn* and *TURBO_window* for other directives

## 27.50 TURBO_optim

| | |
|---|---|
| Syntax | TURBO_optim " [ b \| r \| f ] " |
| Location | Turbo Toolkit v3.00+ |

This is a directive for the TURBO compiler and should be located at the start of your program before any active program lines. The way in which TURBO compiles a program can be optimised using a trade off between speed and code size.

The TURBO_optim directive allows you to dictate how the compiled program is to be optimised and accepts a single character string which should be one of the following values:

- b: Generate BRIEF code, which ensures that the program uses as little memory as possible. This generates the slowest programs.
- r: Optimise code according to REMarks in the program. Normally this will generate BRIEF code unless you include a line containing REMark + in your program which tells TURBO to switch to FAST code. The code will then be optimised for speed until a line containing REMark - is encountered.
- f: Generate FAST code, which ensures that the program runs as quickly as possible. This may however cause the program to need a lot more memory. As with other compiler directives, this value can be changed by configuring the parser_task program or by entering a different value on the Parser's front panel.

**Example**

```
5 TURBO_optim "b"
```

**CROSS-REFERENCE**

See *TURBO_diags*, *TURBO_locstr*, *TURBO_model*, *TURBO_objdat*, *TURBO_objfil*, *TURBO_repfil*, *TURBO_struct*, *TURBO_taskn* and *TURBO_window* for other directives

## 27.51 TURBO_P

| | |
|---|---|
| Syntax | TURBO_P |
| Location | Turbo Toolkit v3.00+ |

This directive forms part of the EXTERNAL and GLOBAL Turbo directives and is used to specify the names of PROCedures contained in another compiled module for a program where that program is loaded as several linked modules (using LINK_LOAD) rather than one huge program.

**NOTE**

Before version 3.00 of the Turbo Toolkit, this directive was called PROCEDURE which would cause problems with installing the Turbo Toolkit under Minerva and SMS.

**CROSS-REFERENCE**

See *TURBO_locstr* and *TURBO_F* for other directives Refer to EXTERNAL for more information. Use *TK_VER$* to check on the version of TURBO toolkit.

## 27.52 TURBO_repfil

| | |
|---|---|
| Syntax | TURBO_repfil filename$ |
| Location | Turbo Toolkit v3.00+ |

This is a directive for the TURBO compiler and should be located at the start of your program before any active program lines. This directive expects you to specify a filename as a string. TURBO will use this file to produce a report on the compilation process, which can be useful to track down compilation errors and warnings. If no filename is specified, then all errors and warnings are merely shown on screen. The full filename (including device) should be specified in quote marks. As with other compiler directives, this value can be changed by configuring the parser_task program or by entering a different value on the Parser's front panel.

**Example**

```
5 TURBO_repfil "ram2_CT_report"
```

**CROSS-REFERENCE**

See *TURBO_diags*, *TURBO_locstr*, *TURBO_model*, *TURBO_objdat*, *TURBO_objfil*, *TURBO_optim*, *TURBO_struct*, *TURBO_taskn* and *TURBO_window* for other directives

## 27.53 TURBO_struct

| | |
|---|---|
| Syntax | TURBO_struct " [ s \| f ] " |
| Location | Turbo Toolkit v3.00+ |

This is a directive for the TURBO compiler and should be located at the start of your program before any active program lines. TURBO is able to compile a wide variety of programs. However, if the program does not follow strict programming rules, it will take longer to compile and will run more slowly (even if TURBO can manage to compile it).

Programs which follow the programming rules are known as Structured. These programming rules are set out below:

1. The main section of the program must appear at the start and not contain any PROCedure or FuNction definitions.
2. At the end of the main section appears only PROCedure and FuNction definitions without any other lines between the end of one definition and start of another except for REMarks.
3. All FOR, REPeat, IF, SELect ON, WHEN, structures are contained within each section (either the main section or a PROCedure / FuNction definition) of the program and not referenced from outside that section.

All other programs are known as Freeform.

The TURBO_struct directive allows you to specify the type of programming style used in the program which is to be compiled. It accepts a single character string which should be one of the following values:

- f: The program is Freeform.
- s: The program is Structured.

As with other compiler directives, this value can be changed by configuring the parser_task program or by entering a different value on the Parser's front panel.

**Example**

```
5 TURBO_struct "s"
```

**CROSS-REFERENCE**

See *TURBO_diags*, *TURBO_locstr*, *TURBO_model*, *TURBO_objdat*, *TURBO_objfil*, *TURBO_optim*, *TURBO_repfil*, *TURBO_taskn* and *TURBO_window* for other directives

## 27.54 TURBO_taskn

| | |
|---|---|
| Syntax | TURBO_taskn name$ |
| Location | Turbo Toolkit v3.00+ |

This is a directive for the TURBO compiler and should be located at the start of your program before any active program lines. This directive allows you to specify the name for the compiled program which will appear in its header and appear when JOBS is used for example. The full name should be specified in quote marks.

As with other compiler directives, this value can be changed by configuring the parser_task program or by entering a different value on the Parser's front panel.

**Example**

```
5 TURBO_taskn "Main v1.2"
```

**CROSS-REFERENCE**

See *TURBO_diags*, *TURBO_locstr*, *TURBO_model*, *TURBO_objdat*, *TURBO_objfil*, *TURBO_optim*, *TURBO_repfil*, *TURBO_struct* and *TURBO_window* for other directives

## 27.55 TURBO_window

| | |
|---|---|
| Syntax | TURBO_window number |
| Location | Turbo Toolkit v3.00+ |

This is a directive for the TURBO compiler and should be located at the start of your program before any active program lines. This tells the TURBO parser to copy across number windows from the existing channel structure into the compiled program. If number=1 only window #1 will appear in the compiled program. Any greater value for number will copy window #0 also. It is usually better to ensure that your compiled program opens all of its own windows, using commands such as:

```
100 OPEN #1,'con_448x200a32x16'
```

This reduces the amount of memory used up by each channel and also ensures that your program only opens the windows which it actually needs to work. As with other compiler directives, this value can be changed by configuring the parser_task program or by entering a different value on the Parser's front panel.

**Example**

```
5 TURBO_window 0
```

**NOTE**

Only the active area of a window is copied across to the compiled program, so if a BORDER has been specified, this will not appear in the compiled program (only the area inside the border will be copied across). If your compiled program then defines its own BORDER on that window, the size of the window will be further reduced.

**CROSS-REFERENCE**

See *TURBO_diags*,*TURBO_locstr*, *TURBO_model*, *TURBO_objdat*, *TURBO_objdat*, *TURBO_objfil*, *TURBO_optim*,*TURBO_repfil*, *TURBO_struct*, and *TURBO_taskn* for other directives

## 27.56 TURN

| | |
|---|---|
| Syntax | TURN [#ch,] degrees |
| Location | QL ROM |

This command is part of the QL's turtle graphics set and alters the current direction of the turtle in the specified window (default #1). When a window is first opened, the turtle will be facing the right hand side of the window (this is zero degrees).

TURN will force the turtle to turn anti-clockwise by the specified number of degrees (note that this does not work in radians!). If a negative number of degrees is specified, the turtle will be turned in a clockwise direction.

**CROSS-REFERENCE**

*TURNTO* forces the turtle to face in an absolute direction. Please also see *MOVE*.

## 27.57 TURNTO

| | |
|---|---|
| Syntax | TURNTO [#ch,] angle |
| Location | QL ROM |

This command, in contrast to TURN forces the turtle in the specified window (default #1) to face in the direction specified by angle. If angle=0, the turtle will face the right hand edge of the window, whereas an angle of 90 will force the turtle to point towards the top of the window. A negative value of angle will cause the turtle to turn clockwise, so that angle=-90 is the same as angle=270.

**CROSS-REFERENCE**

Please refer to *MOVE* and *TURN*.

## 27.58 TXTRAS

| | |
|---|---|
| Syntax | TXTRAS [#ch] |
| Location | TinyToolkit |

This command lists extensions to SuperBASIC in the specified channel #ch (default #1). Apart from printing the mere keyword name, it will also report the type, ie. whether it is a function or command.

**Example**

TXTRAS might print:

```
Proc RUN
Proc STOP
Proc OPEN
Proc CLOSE
```

**NOTE**

On pre 1.10 versions of TinyToolkit, TXTRAS was named EXTRAS.

**CROSS-REFERENCE**

*EXTRAS*, *SXTRAS* and *VOCAB* are all similar.

## 27.59 TYPE

| | |
|---|---|
| Syntax | TYPE (name$) |
| Location | TinyToolkit, BTool |

The function TYPE returns the internal identification number of any variable, device name, keyword, command, function etc. as a decimal number. Each type corresponds to a certain number:

| Hex | Dec | Type | Example |
|---|---|---|---|
| 0001 | 1 | undefined string | Name$ |
| 0002 | 2 | undefined floating point | Size |
| 0003 | 3 | undefined integer | Age% |
| 0201 | 513 | string variable | Name$="Smith" |
| 0202 | 514 | floating point number | Size=1.85 |
| 0203 | 515 | integer number | Age%=38 |
| 0301 | 769 | string array | DIM a$(10,20) |
| 0302 | 770 | floating point array | DIM a(221) |
| 0303 | 771 | integer array | DIM a%(10000) |
| 0400 | 1024 | BASIC PROCedure | DEFine PROCedure QUIT |
| 0501 | 1281 | BASIC string FuNction | DEFine FuNction Who$ |
| 0502 | 1282 | BASIC floating point FuNction | DEFine FuNction Tm(day) |
| 0503 | 1283 | BASIC integer FuNction | DEFine FuNction Age% |
| 0602 | 1538 | REPeat loop name | REPeat forever |
| 0702 | 1794 | FOR loop name | FOR i=1 TO n |
| 0800 | 2048 | machine code procedure | RUN, ED, NEW |
| 0900 | 2304 | machine code function | QDOS$, VER$, FILL$ |

**NOTE 1**

Parameters must be given in quotes if you want to find out the type of the actual name, eg:

```
PRINT TYPE ('RUN')
```

If quote marks are not used, then the value of the parameter is passed instead - eg:

```
name$="RUN"
PRINT TYPE(name$)
```

will not return the type of name$ but the type of RUN.

**NOTE 2**

TYPE can also take any kind of expression, whether or not they are valid.

**CROSS-REFERENCE**

*KEY_ADD* and *ELIS* return the address where a machine code keyword is stored. *DEFINED* checks if a variable is set.

## 27.60 TYPE_IN

| | |
|---|---|
| Syntax | TYPE_IN string$ |
| Location | BTool |

Same as FORCE_TYPE.

## 27.61 T_COUNT

| | |
|---|---|
| Syntax | T_COUNT [ (watch) ] |
| Location | Timings (DIY Toolkit - Vol H) |

This function reads the time elapsed on the specified stop- watch (default 1). If the watch has not been started, the value 2,147,483,647 is returned by this function.

**CROSS-REFERENCE**

See *T_START* and *T_STOP*. *T_ON* contains a general description of the stop-watches.

## 27.62 T_OFF

| | |
|---|---|
| Syntax | T_OFF |
| Location | Timings (DIY Toolkit - Vol H) |

This command removes all of the stop-watches from memory, although they can be re-enabled with T_ON.

**NOTE**

None of the times on the stop-watches are reset and can therefore be continued once T_ON has been used.

**CROSS-REFERENCE**

See *T_ON*.

## 27.63 T_ON

| | |
|---|---|
| Syntax | T_ON |
| Location | Timings (DIY Toolkit - Vol H) |

This toolkit provides the QL with five independent stop-watches which can be used to make accurate timings (more accurate than using DATE).

The stop-watches are linked into the QL's 'polled list' of small routines which are run every frame on the computer (1/50 second on a British QL, 1/60 on most foreign QLs). There is a slight disadvantage in using the polled interrupts in that they are sometimes disabled by machine code routines, for example when accessing microdrives and disks.

Because of this, these commands are not much for timing programs which depend heavily on external hardware. These stop-watches are however very useful for comparing the speed of various program routines without having to make thousands of loops in order to show any difference in speed.

This command enables all the stop-watches. This must be issued before T_START can be used.

**CROSS-REFERENCE**

See *TTV*, *T_START*, *T_STOP*, *T_RESTART*

## 27.64 T_RESTART

| | |
|---|---|
| Syntax | T_RESTART [watch] |
| Location | Timings (DIY Toolkit - Vol H) |

This command restarts a specified stop-watch (default 1) once it has been stopped, without resetting the initial time to zero. This command can have spurious effects if the stop-watch has not previously been used.

**CROSS-REFERENCE**

*T_STOP* stops a stop watch. See *T_START* also.

## 27.65 T_START

| | |
|---|---|
| Syntax | T_START [watch] |
| Location | Timings (DIY Toolkit - Vol H) |

This command starts the specified stop-watch (default 1), setting the initial time to zero.

**CROSS-REFERENCE**

You need to have used *T_ON* before *T_START* can be used. See also *T_STOP* and *T_RESTART*.

## 27.66 T_STOP

| | |
|---|---|
| Syntax | T_STOP [watch] |
| Location | Timings (DIY Toolkit - Vol H) |

This command stops the specified stop-watch (default 1) from running.

**CROSS-REFERENCE**

*T_RESTART* restarts a stop-watch. *T_START* starts a stop-watch from afresh.

CHAPTER

# TWENTYEIGHT

# KEYWORDS U

## 28.1 UINT

| | |
|---|---|
| Syntax | UINT (x%) |
| Location | BTool |

The function UINT returns the unsigned value of a (signed) integer:

```
unsigned = signed% + 2^16
```

or:

```
unsigned = UINT(signed%)
```

**CROSS-REFERENCE**

*SINT*

## 28.2 UNDER

| | |
|---|---|
| Syntax | UNDER [#ch,] switch |
| Location | QL ROM |

This command switches underlining in the specified window (default #1) either on or off. Underlining is enabled if switch=1 or disabled if switch=0. Other values of switch will return a 'bad parameter' error.

If underlining is enabled, whenever anything is PRINTed, a line will be drawn in the current INK colour in the bottom but one row of the character.

If FLASH is enabled, although the character will continue to flash, the underline itself will not. MODE will disable underlining.

**Example 1**

```
UNDER 1: PRINT 'Title:'!: UNDER 0: PRINT !'QL SuperBASIC'
```

**Example 2**

If you don't like the line which is drawn by underline than you can use OVER to draw your own line with a different colour. Note however that this line ought to be drawn before the underlined text since the line should not overlap letters like g, p, q and j.

```
100 DEFine PROCedure PRNT_UNDL (ch, x, y, text$, col1, col2)
110   AT#ch,x,y: INK#ch,col2: OVER#ch,0
120   PRINT#ch,FILL$("_",LEN(text$))
130   AT#ch,x,y: INK#ch,col1: OVER#ch,1
140   PRINT#ch,text$
150   OVER#ch,0
160 END DEFine PRNT_UNDL
```

```
PAPER 3: CLS
PRNT_UNDL #1,3,3,"Looking good.",7,0
```

**NOTE**

MODE will reset the current underline mode in all windows.

**CROSS-REFERENCE**

*INK* sets the current ink colour for the specified channel, *PRINT* prints out characters.

## 28.3 UNJOB

| | |
|---|---|
| Syntax | UNJOB drive_filename |
| Location | UNJOB |

This command sets the file type of the given file (the full filename must be supplied) to zero. The reason for this command is that certain assemblers and tools set the file type to 1 (executable file) even though the file cannot be started as a job. Since commands like EX or EXEC check the file type to decide whether a file can be executed, they will try to start such a file and crash the system in most cases. A simple UNJOB prevents this in the long term.

**NOTE**

v1.00 of this command did not work on most QL ROM versions, giving a bad parameter error.

**CROSS-REFERENCE**

Each file has a file type which can be found with the *FTYP* and *FILE_TYPE* functions or directly by looking at the file header (*HEADR*). It is also possible to set the file type by rewriting the whole file header with *HEADS*; alternatively, *UNJOB* does the same.

## 28.4 UNL

| | |
|---|---|
| Syntax | UNL |
| Location | Beuletools |

This function returns the control codes needed to switch on underline printing on an EPSON compatible printer, PRINT UNL is the same as:

```
PRINT CHR$(27) & "-" & CHR$(1)
```

**CROSS-REFERENCE**

*NORM*, *BLD*, *EL*, *DBL*, *ENL*, *PRO*, *SI*, *NRM*, *ALT*, *ESC*, *FF*, *LMAR*, *RMAR*, *PAGDIS*, *PAGLEN*.

## 28.5 UNLOAD

| | |
|---|---|
| Syntax | UNLOAD program_name |
| Location | MutiBASIC (DIY Toolkit - Vol M) |

Despite the name, this toolkit is completely different to the MultiBASICs which are provided on Minerva ROMs. This toolkit actually provides a quick means of saving and loading programs in memory - this allows you to load a program which you are working on, store it in memory and then alter the program. If the new alterations to the program do not work out as planned and you want to revert to the original version, you can simply RELOAD the original version from program in a matter of seconds (rather than the minutes which it would take to LOAD the original version from disk).

This can be very useful for program development, or, for example, if you have a SuperBASIC utility program which you use a lot.

This command allows you to store the currently loaded SuperBASIC program in memory. You have to supply a name for the program (similar to the name which you could use with the SAVE command, except there is no need for a device name and the program name can be up to 127 characters long). The program is then stored - details of the programs which have been stored with this command are available from the jobs list (see JOBS). When the program is stored in memory, the contents of all variables and pointers are also stored, which makes certain that if you UNLOAD a program whilst it is RUNning, you can later RELOAD it and re-start it from the same place (with CONTINUE).

Version 4.0+ of the toolkit, allows you to store the current screen display and mode along with the program, so that when the program is RELOADed, the display is in a known layout. To further extend the usefulness of this toolkit, any commands which appear after UNLOAD will be automatically executed when the program is RELOADed, for example:

```
UNLOAD test: RUN
```

will always RUN the program when you:

```
RELOAD test
```

**NOTE 1**

The toolkit expects the display to be located at 131072 and be 512x256 pixels and so you should switch off the screen storage facility if you are using a higher resolution display or a dual screen system.

**NOTE 2**

If a job already exists with the name which you have given to the program, 'Already Exists' will be reported.

**NOTE 3**

Although the toolkit can be used to store programs from a Minerva MultiBASIC, you cannot load the toolkit from a Multiple BASIC - an 'incomplete' error is reported.

**NOTE 4**

The current channel details are not stored when you use UNLOAD - you may therefore need to re-open the channels when the program is RELOADed, or use something akin to:

```
UNLOAD 'watch': OPEN #3,con_448x200a32x16
```

which will always ensure that #3 is OPEN whenever the program is RELOADed.

**NOTE 5**

If a program uses ALCHP to grab some memory, unless you intend to always RUN the program from the start when you RELOAD it, do not use any command which will release this area of common heap memory before you RELOAD the program. Commands which do this include:

```
CLCHP
CLEAR
NEW
LOAD
```

**WARNING 1**

This toolkit does not work on SMSQ/E and can crash the computer.

**WARNING 2**

Unfortunately, attempts to use this toolkit to UNLOAD files from one interpreter and then RELOAD the files into another Multiple BASIC will crash that Multiple BASIC (or have other various undesirable effects).

**CROSS-REFERENCE**

*SCR_SAVE* allows you to dictate whether the screen display and mode should be stored together with the program. *RESAVE* is similar. *REMOVE* allows you to remove a program stored in memory with this command. See also *RELOAD* and *QSAVE*.

## 28.6 UNLOCK

| | |
|---|---|
| Syntax | UNLOCK file,code$,code |
| Location | CRYPTAGE |

See *LOCK*.

**Example**

```
UNLOCK ram1_secret_txt,"Phew",7241
```

## 28.7 UNSET

| | |
|---|---|
| Syntax | UNSET (variable) |
| Location | PARAMS (DIY Toolkit - Vol P) |

This is the same as *DEFINED* and suffers from the same problem!

## 28.8 UPC$

| | |
|---|---|
| Syntax | UPC$ (string$) |
| Location | LWCUPC |

This is the same as *UPPER$*.

## 28.9 UPPER$

| | |
|---|---|
| Syntax | UPPER$ (string$) |
| Location | TinyToolkit, Function (DIY Toolkit - Vol R) |

This function takes the given string and converts any lower case letters to capitals and then returns the whole string. Normally, only the ASCII alphabet is catered for, which means that no national characters are converted, ie. the function only works with A..Z and a..z.

The DIY Toolkit version will cope with accented characters, but you may have to modify the source code in order for this function to work with some international character sets which use an extended alphabet.

**Example**

This is not quite an example for UPPER$ but a replacement which converts all characters where an upper character is available:

```
100 DEFine FuNction UPPER_$ (string$)
110   LOCal i,c,u,u$: u$=""
120   FOR i=1 TO LEN(string$)
130   c=CODE(string$(i)): u=c
140   SELect ON c=97 TO 122: u=c-32:=128 TO 139: u=c+32
150   u$=u$ & CHR$(u)
160   END FOR i
170   RETurn u$
180 END DEFine UPPER_$
```

**CROSS-REFERENCE**

*UPC$* returns the same as *UPPER$*. See also *ConvCASE$* and *LOWER$*.

## 28.10 UPUT

| | |
|---|---|
| Syntax | UPUT [#ch\position,] [item *[,item$^i$]* ..] or<br>UPUT [#ch,] [item *[,item$^i$]* ..] |
| Location | SMSQ/E v2.55+ |

This command is the same as BPUT, except that any bytes sent by it to the specified channel (default #3) are not affected by the TRA command. This command is therefore useful for sending printer control codes.

**CROSS-REFERENCE**

See *BPUT*, *WPUT* and *LPUT*.

## 28.11 USE

| | |
|---|---|
| Syntax | USE [ #channel ] |
| Location | USE (DIY Toolkit - Vol C) |

Many commands and functions which are described in this manual, expect a channel number to be passed to them and if one is not supplied, will default to a specific channel. This command can be used to re-direct all machine code commands and functions which normally default to #1.

After using this command, if a channel parameter is not specified, the commands and functions will then default to the channel specified by USE instead of #1. Also, *even* if you explicitly pass a channel number #1 as a parameter to a command or function, then the command or function will *still* be re-directed to the channel specified by USE. If no parameter is specified, then this is equivalent to USE #1.

**Example**

```
PRINT 'This is channel #1': USE #2: PRINT 'This is using Channel #2'
PRINT #1, 'This is still channel #2' USE: PRINT 'This is channel #1 again!'
```

**NOTE**

There is a slight difficulty in using this command in that when you USE #1 some of the information used by SuperBASIC for the channel which you have been using as the default will be lost. This is the last graphics co-ordinates, turtle graphics direction, pen status, character position on line and line width for files (set with WIDTH) will be lost. You will also lose the original values for these offsets for channel #1 (ie. the values which were in use prior to the USE #ch command). Instead, the values are set to pen up, position 0,0, width 80, direction left-to-right. You can use:

```
PEEK_W(\48\chan*40+offset)
```

to store these values before the USE call and then restore them with POKE. Refer to QDOS/SMS Reference Manual Section 18.4.1 to find out how these values are stored.

**WARNING**

If used from within a SMS SBASIC, v0.2 (at least) of this command will crash the computer when output is redirected to #1 using either USE or USE#1. The problem only occurs when you try to send output to #1.

**CROSS-REFERENCE**

*PRINT*,*CSIZE*,*INK*,*PAPER* and*STRIP* are just a few of the commands which default to #1 and are therefore affected by this command.

## 28.12 USE_FONT

| | |
|---|---|
| Syntax | USE_FONT #channel, font1_address, font2_address |
| Location | DJToolkit 1.16 |

This is a procedure that will allow your programs to use a character set that is different from the standard QL fonts. The following example will suffice as a full description.

**EXAMPLE**

```
1000 REMark Change the character set for channel #1
1010 :
1020 REMark Reserve space for the font file
1030 size = FILE_LENGTH('flp1_font_file')
1040 IF size < 0
1050    PRINT 'Font file error ' & size
1060    STOP
1070 END IF
1080 :
1090 REMark Reserve space to load font into
1200 font_address = RESERVE_HEAP(size)
1210 IF font_address < 0
1220    PRINT 'Heap error ' & font_address
1230    STOP
1240 END IF
1250 :
1260 REMark Load the font
1270 LBYTES flp1_font_file, font_address
1280 :
1290 REMark Now use the new font
1300 USE_FONT #1, font_address, 0

.......Rest of program

9000 REMark Reset channel #1 fonts
9010 USE_FONT #1, 0, 0
9020 :
9030 REMark Release the storage space
9040 RELEASE_HEAP font_address
```

CHAPTER

# TWENTYNINE

# KEYWORDS V

## 29.1 VA

| | |
|---|---|
| Syntax | VA (i,n) |
| Location | Toolfin |

The function VA returns the value of $(1+i)^{-n}$ = 1/MT(i,n) where i and n can be any floating point numbers (see MT for error handling).

**Example**

VA allows you to find out about base capital which will grow to a certain higher (i>0) capital at the interest rate i over n periods. Assume that you want to buy an expensive car for $80000 in two years and your investment returns an annual gain of 10% (not bad), then you need to invest 80000 * VA(1/10, 2) = 66115.7

**CROSS-REFERENCE**

You can check the result of the above example with: 66115.7 * *MT*(1/10,2) = 80000.

See also *VFR*, *VAR*, *TCA*, *TNC*, *TEE*, *RAE*, *RAFE*.

## 29.2 VAR

| | |
|---|---|
| Syntax | VAR (i,n) |
| Location | Toolfin |

The function VAR returns the value of: $((1+i)^n-1) / (i* (1+i)^n)$ where i and n can be any floating point numbers (see MT for error handling).

**CROSS-REFERENCE**

*MT*, *VA*, *VFR*, *TCA*, *TNC*, *TEE*, *RAE*, *RAFE*

## 29.3 VER$

| | |
|---|---|
| Syntax | VER$ or<br>VER$ [(n)] with n=1, 0, -1, -2(Minerva and SMS only) |
| Location | QL ROM |

The function VER$, which is the same as VER$(0) returns a short identification code for the version of the current system ROM. Here are most of the possible values (in order of development):

**Original ROMs**

| VER$ | Explanation |
|---|---|
| FB | This is the first ROM sold in April 1984, QDOS version 1.00. It comes with a 'Dongle' - a board which needed to be plugged into the QL's ROM port. It is very unreliable and should be replaced!! |
| PM<br>EL<br>TB | These three ROMs were developed during the following two months. (May, June & July 1984.) |
| AH | Released as the "definitive" version in June 1984. |
| JM | British QL, QDOS v1.03, the first version which was exported. |
| JS | Released in spring 1985, QDOS v1.10. Also found on early Thors and patched in ST/QL and early Amiga Emulators. |
| JSU | American QL. |
| The following versions were only sold in their respective countries. All are QDOS v1.03. | |

continues on next page

Table 1 – continued from previous page

| VER$ | Explanation |
|---|---|
| MGD | Danish |
| MGE | Spanish |
| MGF | French<br>German<br>Italian<br>Norwegian |
| MGG | Swedish |
| MGI | Swedish<br>Finish |
| MGN | Greek<br>Greek |
| MGS | |
| MGB | |
| MGY | |
| MG$ | |
| $FP | |

**Patches**

| VER$ | Explanation |
|---|---|
| MGUK | A version of the MGx ROM produced independently for the UK market. |
| MG | Another patched version, mainly distributed in Germany. |
| MGUS | Out of the three patches, this is the only legal one and was produced for the United States. |

**New developments**

| VER$ | Explanation |
|---|---|
| CS<br>PT<br>PO | Different ROMs used on the THOR XVI. |
| JSL1 | QL with Minerva ROM, a very much debugged and enhanced version of the JS ROM, available in all languages for all kinds of QLs. |
| HBA | Either the SMSQ or SMSQ/E replacement operating system for QXLs, Atari ST/STE and TT series computers and the Miracle Gold Card family of add-on cards. |

**NOTE 1**

VER$ can be used to write flexible programs which adapt themselves to specific features of computers and ROM implementations. However, if you intend to test VER$, for example:

```
IF VER$= 'JM'
```

to retain compatibility with the different ROM versions, you must first assign the contents of VER$ to a variable:

```
100 a$=VER$
110 IF a$(1 TO 2)='MG': PRINT 'MG ROM'
```

**NOTE 2**

The names of the original ROMs were derived from names of Clive Sinclair's secretaries, taxi drivers he met and so on. (Just in case you are looking for any sense behind the abbreviations.)

**MINERVA NOTES**

On Minerva, VER$ accepts a parameter:

- VER$(0)as per above VER$.
- VER$(-2) returns the base address of the system variables (normally $28000 = 163840 on a standard QL).
- VER$(-1)returns the current job identification number.
- VER$(1)returns the version of QDOS (see also QDOS$).

**SMS NOTES**

VER$ has been amended to provide the same facilities as on Minerva.

**WARNING**

If you fail to assign VER$ to a variable before testing its value, then you can crash a JS (or JSU) ROM. This will also happen on Minerva ROMs (pre v1.77) with Minerva's extended variant VER$(n).

**CROSS-REFERENCE**

*QDOS$* returns the version number of QDOS in the same way as *VER$*(1). See also *MACHINE* and *PROCESSOR*.

## 29.4 VFR

| | |
|---|---|
| Syntax | VFR (i,n) |
| Location | Toolfin |

The function VFR returns the value of: $((1+i)^n-1)/i$ where i and n can be any floating point numbers (see MT for error handling).

**CROSS-REFERENCE**

*MT*, *VA*, *VAR*, *TCA*, *TNC*, *TEE*, *RAE*, *RAFE*

## 29.5 VG_HOCH

| | |
|---|---|
| Syntax | VG_HOCH (fontnr) fontnr=0..15 |
| Location | BGI |

This function returns the maximum height of the specified font (fontnr=0..15) in pixels if printed with the current size settings of VG_PARA.

**CROSS-REFERENCE**

*VG_PARA* and *VG_LOAD*.

## 29.6 VG_LOAD

| | |
|---|---|
| Syntax | VG_LOAD fontnr, file$ fontnr=0..15 |
| Location | BGI |

This toolkit allows the QL to use BGI vector fonts (common on the PC) to draw on the screen. There are now numerous fonts available in this format for the QL, being the same format as used by the PROWESS programming system from PROGS.

The advantage of vector fonts is that they can be drawn on screen at any size and angle without affecting the legibility. Each character is not made up by a bit-map (as with the original QL fonts), but by a description of how each line is drawn to make up a character.

This command forces a BGI font file$ to be loaded from a file into memory. Fontnr is the number of the font Up to 16 fonts can be loaded at the same time; Fontnr may range from 0 to 15. The file$ can be any font in standard BGI format, for example those which are delivered with Turbo Pascal and Turbo C by Borland. The format used on the QL with this Toolkit is binary compatible.

If VG_LOAD fails to load a file for external reasons (eg. if the file is not found), the font which was previously attached to fontnr will have been lost.

**Example**

```
VG_LOAD 1, "flp1_goth_chr"
```

**WARNING**

You have to ensure that file is actually a BGI font. Otherwise your machine will almost surely crash.

**CROSS-REFERENCE**

See *VG_PRINT* about displaying text using a vector font and the other VG_XXX commands.

## 29.7 VG_PARA

| | |
|---|---|
| Syntax | VG_PARA col, xsize, ysize, angle, qlibm, italic, bold |
| Location | BGI |

The use of the command VG_PARA is easier than the large parameter list may suggest. VG_PARA specifies how text should look when printed with VG_PRINT. The colour col does not allow strips and

textures, col may only range from 0 to 7, other values are modulated appropriately. xsize and ysize determine the size of the font (not in pixels!), they can be freely chosen from any non-negative values, but sizes smaller than three are usually not readable.

Angle is the angle (0..359º) by which the text should be rotated. This is different from italics because the angle parameter rotates the text around the origin point of the text whilst italics slopes each character. The effect of italics is not linear, values between -10 and 10 give all kinds of slope; negative italics slope to the left and positive to the right.

The effect of bold on the other hand is easily described: bold refers to the thickness of the characters' lines which are bold+1 pixels.

Qlibm is a switch: any non-negative value will make VG_PRINT try to find the character which matches best to the one given in the text to be printed; this works for IBM fonts only, see VG_PRINT for further explanation of this point. The default setting is VG_PARA 7,8,8,0,0,0,0 ie. white colour, 8x8 size, no italics, bold, rotation or conversion.

**Examples**

Both examples assume a BGI font loaded to font number 0 and the default VG_WIND settings (VG_WIND 0,511,0,255). The screen should be emptied with:

```
WIPE
```

or:

```
WINDOW 512,256,0,0: CLS
```

```
100 FOR size=1 TO 25
110   bold = (size=25)
120   VG_PARA 5.5*size/25,size,size,0,0,-3,bold
130   VG_PRINT 70-2*size,150-size,0,"Sinclair QL"
140 END FOR size
```

```
100 FOR angle=0 TO 3000 STEP 12
110   xsize=4*SIN(RAD(angle))+8
120   VG_PARA 7,xsize,10,angle,0,0,0
130   VG_PRINT 200,120,0,"Yippie"
140   VG_PARA RND(0 TO 2),xsize,10,angle,0,0,0
150   VG_PRINT 200,120,0,"Yippie"
160 END FOR angle
```

**WARNING**

A negative bold parameter will cause VG_PRINT to fall into an infinite loop. This hangs the job which called VG_PRINT indefinitely.

**NOTE**

Negative sizes lead to strange output but do no harm. BGI fonts come in different sizes so that the size settings of VG_PARA do not necessarily reflect the actual size that text will be; check with VG_HOCH for every font.

**CROSS-REFERENCE**

*VG_HOCH* is a function which returns the text sizes.

## 29.8 VG_PRINT

| | |
|---|---|
| Syntax | VG_PRINT x, y, fontnr, text$ |
| Location | BGI |

The command VG_PRINT prints text$ at the absolute position x, y on the screen. The font which has been applied to fontnr with VG_LOAD will be used; there will be no output if the font number has not been used yet. If x and y are not inside the area defined with VG_WIND or the text is too high to fit, again there will be no output. VG_PRINT works only in high resolution mode (MODE 4).

**Example**

```
VG_LOAD 1,flp1_goth_chr
VG_WIND 0,511,0,255
VG_PRINT 100,100,1,"Hello World"
```

**NOTE**

Since the BGI fonts will usually originate from another computer system which uses a different character set, text$ and the actual output may differ dramatically if text$ contains characters which are not standardised in ASCII, especially national characters (umlauts, acutes). The following program lists the complete character set of a BGI font:

```
100 VG_LOAD 0,"flp1_goth_chr"
110 VG_WIND 0,511,0,255
120 WINDOW 512,256,0,0: PAPER 0: CLS
130 VG_PARA 7,5,5,0,0,0,0
140 FOR c=0 TO 255
150   VG_PRINT 20*(c MOD 20)+50,20*(c DIV 20),0,CHR$(c)
160 END FOR c
```

**CROSS-REFERENCE**

*VG_PARA* offers a switch to approximate an identity between text$ and display as far as possible. Modify line 130 in the above listing so that it looks like this now:

```
130 VG_PARA 7,5,5,0,1,0,0
```

Running the program again shows you *VG_PRINT*'s attempts to correct the problem. See also the other VG_XXX keywords!

## 29.9 VG_RESO

| | |
|---|---|
| Syntax | VG_RESO scradr, xres, yres |
| Location | BGI |

This command defines the screen base address and the screen size for printing the BGI fonts. The default is:

```
VG_RESO 131072,512,256
```

This will need to be changed if the screen offset is not 131072. A fixed screen address should never be assumed. VG_RESO can also be used to write to the second screen on a dual screen system.

**CROSS-REFERENCE**

The settings of *VG_WIND* are dependent on *VG_RESO*'s. See *SCRBASE*, *PEEK* and *MODE*.

## 29.10 VG_WIND

| | |
|---|---|
| Syntax | VG_WIND x1, x2, y1, y2 |
| Location | BGI |

This command defines a rectangular area of the screen. Only text printed inside this window with VG_PRINT (it's not a window in SuperBASIC terms) will be visible:

```
  0                         511
0 +-------------------------+
  |                         |
  |   x1        x2          |
  | y1+---------+           |
  |   |         |           |
  |   |BGI Text |           |
  |   |         |           |
  | y2+---------+           |
55+-------------------------+
```

It is strongly recommended that you specify a window inside the physical screen, so: 0 <= x1 < x2 <= 511 and 0 <= y1 < y2 <= 255 (assuming a standard 512x256 pixel screen - replace the upper bounds if you have a better graphics card, eg. QVME). Note that VG_WIND does not check the parameters (this is impossible without the Window Manager).

Default settings are x1=0, x2=511, y1=0, y2=255.

**Example**

```
VG_WIND 0,511,0,255
```

restores these defaults.

**CROSS-REFERENCE**

See also the other VG_XXX keywords. *QFLIM* can be used to find out about about the size of the screen if the Window Manager is loaded.

## 29.11 VIEW

| | |
|---|---|
| Syntax | VIEW [#channel,] text_file or<br>VIEW \channel, text_file |
| Location | Toolkit II, THOR XVI |

This command reads the contents of the given text_file line by line and prints it to the given channel (default #1). If a line is longer than the window, it is not split and continued in the next line (as PRINT would do) but truncated. The second syntax allows you to open a temporary channel to which the output will be sent, for example you could use:

```
VIEW \con,text_file
```

or:

```
VIEW \ram1_test,flp1_text_file
```

Note that the latter is the same as:

```
COPY flp1_text_file TO ram1_test
```

Lines in a text file are separated by line feed characters <LF>, ie. CHR$(10). If output is sent to a window, then when a window page is full, VIEW generates a <CTRL><F5>, and waits for a keypress to continue VIEWing. Sub-directories and default directories are supported by this command, which will look on the default data directory for the given file if necessary (see DATAD$).

**NOTE**

If the final line in the file being VIEWed does not contain a line feed, it will not appear on screen.

**WARNING**

There is a possibility that if a file is longer than 32767 characters and does not include a newline character, the system may crash!

**CROSS-REFERENCE**

*SPL* file TO #1 copies all kinds of files to a window, for example without truncating lines.

Compare *COPY* and *MORE*.

## 29.12 VOCAB

| | |
|---|---|
| Syntax | VOCAB [ #channel, ] type or<br>VOCAB [ #channel ] [,type] |
| Location | VOCAB (DIY Toolkit, Vol X) |

This command lists all of the names which fall into a given category and are recognised by the SuperBASIC interpreter in the given channel (default #1). If type is not specified, then it is assumed to be type=8.

The names are listed in columns, calculated by reference to the width of the specified channel (set by WIDTH for non-window devices). Once all of the names of the given type have been listed, a line feed is printed to end the list. The values for type are:

| Type | Category of names listed |
|---|---|
| 0 | Unset Names |
| 2 | Simple Variables |
| 3 | Dimensioned Variables |
| 4 | SuperBASIC PROCedures |
| 5 | SuperBASIC FuNctions |
| 6 | Used REPeat loops |
| 7 | Used FOR loops |
| 8 | Machine code Procedures |
| 9 | Machine code Functions |

Other values or type=1 will report an error or may cause junk to appear on screen.

**NOTE 1**

For some reason, under SMS, VOCAB 2 will report rubbish on screen unless a program has been RUN already and even then, the last entry may not be an actual variable. No such problems seem to occur on Minerva or other ROMs.

**NOTE 2**

VOCAB 6 and VOCAB 7 only list those REPeat and FOR loop names which have actually been used in the program when it has been RUN.

**CROSS-REFERENCE**

Use *SXTRAS* if you have a lot of extensions in memory and you are looking for a specific one.

See also *TXTRAS*, *EXTRAS* and *TYPE*.

CHAPTER

THIRTY

# KEYWORDS W

## 30.1 WAIT_EVENT

| | |
|---|---|
| Syntax | WAIT_EVENT (event [,timeout] ) |
| Location | SMSQ/E v2.71+ |

This function access the Event Accumulator for the current job and checks whether the specified event (or events) have occurred. If you want to check for the occurrence of several events, you merely need to add together the numbers of the events. If any one of the specified events has already occurred then the function will return the total value of the specified events which have occurred. If none of the specified events have occurred, then this function will suspend the current program until one of those events has occurred or the specified timeout (if any) has elapsed. If timeout is not specified then the function will wait forever. If the reason for the function returning was that the timeout has elapsed (and none of the specified events have occurred) then the returned value will be 0.

**Example**

```
PRINT WAIT_EVENT(12)
```

This wait for event numbers 4 and 8 (4+8=12). If event 8 was notified as having occurred, then the value 8 would be shown on screen.

**CROSS-REFERENCE**

*SEND_EVENT* notifies a Job's Event Accumulator that one or more events have occurred.

## 30.2 WBASE

| | |
|---|---|
| Syntax | WBASE [(#channel)] |
| Location | Tiny Toolkit |

This function is exactly the same as WIN_BASE.

**CROSS-REFERENCE**

See *WIN_BASE*.

## 30.3 WCOPY

| | |
|---|---|
| Syntax | WCOPY [#ch,] [wild1] [TO wild2](Toolkit II) or<br>WCOPY [#ch,] wild1 TO wild2(THOR) |
| Location | Toolkit II, THORs |

The command WCOPY is intended to allow you to copy several files with a common root from one device to another, quickly and easily.

It is however necessary to understand the way in which Toolkit II's wildcards work, as WCOPY uses these wildcards to find the required files. A wildcard is a means of finding several files which have similar names.

The first thing which any wildcard command does is to look at the supplied parameter and then compare this against each entry in the directory of the given device. If any of the filenames match exactly, or if the parameter forms the start of any filenames, those files are marked as chosen. For example:

```
WCOPY flp1_D TO flp2_
```

would copy all files whose names are either 'D' or begin with the letter 'D' to flp2_ (the comparison is case independent).

However, wildcards can be much more complex and wonderful on the QL. If you place two underscores ('_') together as part of wild1, this is taken to be a wildcard and can in fact be replaced by any string of characters in order to match filenames with wild1. Wildcards are further complicated by the fact that if a device name is not provided as part of wild1, then the default device will be added (which ends with an underscore, so if wild1 begins with an underscore, you will have a wildcard symbol!!) A few examples of wildcards (assuming default device is 'flp1_'):

| Wild1 | Wild Card Name | File Matches |
|---|---|---|
| t | flp1_t | flp1_testa<br>flp1_test_v1.00_bas |
| _t | flp1__t | flp1_testa<br>flp1_test_v1.00_bas<br>flp1_old_v0.01_test_bas |
| flp1_old_ | _flp1_old__ | flp1_old_v0.01_test_bas<br>flp1_old_v1.00_exe<br>flp1_old_data |

WCOPY uses both wildcards for ascertaining the names of the files to be copied, and the files to be created. However, both wild1 and wild2 are dealt with distinctively.

WCOPY will use the rules on wildcards to search for files which match with wild1 on the specified device, or the default data device if no device is specified. However, the rules for determining the destination parameter wild2 are complex:

1. If no device is given, but a filename is specified, WCOPY looks at wild1. The destination device is then assumed to be the same as the source device (ie. the device name specified as part of wild1, or if omitted, DATAD$).
2. If the second parameter is omitted, then again WCOPY looks at wild1. If a device is given in the first parameter, then this is used as the destination device. On the other hand, if no device was specified, then the default destination device will be used (see DESTD$).
3. If a second parameter is given which includes a device name, then this is used! Having decided upon the device to which the files are to be copied, WCOPY then looks at the remainder of wild2 to ascertain what to do with the filenames it has found.

Before trying to understand how this works, it is essential to realise that there is an implicit wildcard placed at the end of both wild1 and wild2.

WCOPY will look at wild2 and compare each filename that it has found using the wildcards in wild1 in turn. If a wildcard in wild1 is matched by a wildcard in wild2, then that part of the source filename will be inserted into the destination filename. However, beyond this, WCOPY will use the rest of wild2 as the actual destination filename. Any additional sections in wild1 or wild2 will be inserted after the drive name in the destination filename. See the examples below!

Having decided which files are to be copied and the names they are to be given on the device where they are being copied to, WCOPY will then request confirmation in the specified channel (default #0) for each file, by printing the following message in the channel:

```
source_file TO destination_file..Y/N/A/Q?
```

You will then need to press <Y> to copy that file across, <N> to miss that file out, <A> to copy all files which match with wild1, or <Q> to leave WCOPY. In this instance, <ESC> and <CTRL><SPACE> both act as <Q>.

If the destination file already exists, another prompt will be shown in the form:

```
OK to overwrite..Y/N/A/Q?
```

You will then need to press <Y> to overwrite that file, <N> to go onto the next file, <A> to overwrite this and all other files being copied if they already exist, or <Q> to stop WCOPY. Again, <ESC> and <CTRL><SPACE> act as <Q>.

**Examples**

Assuming that the default data device is flp1_ and the default destination device is ram2_:

```
WCOPY
```

Copies all files on flp1_ to ram2_

```
WCOPY flp1_test TO ram2_old
```

Copies:

```
flp1_testa to ram2_olda
flp1_test_v1.00_bas to ram2_old_v1.00_bas
```

```
WCOPY flp1_test, ram2_old_
```

Copies:

```
flp1_testa to ram2_olda
flp1_test_v1.00_bas to ram2_old_v1.00_bas
```

```
WCOPY _bas to ram2_
```

Copies:

```
ram1_test_v1.00_bas to ram2_bas
ram1_old_v0.01_test_bas to ram2_bas
```

```
WCOPY _bas, ram2__
```

Copies:

```
ram1_test_v1.00_bas to ram2_test_v1.00_bas
ram1_old_v0.01_test_bas to ram2_old_v0.01_test_bas
```

```
WCOPY old__ TO ram2_
```

Copies:

```
ram1_old_v0.01_test_bas to ram2_v0.01_test_bas
ram1_old_v1.00_exe to ram2_v1.00_exe
ram1_old_data to ram2_data
```

**NOTE 1**

The TO in the syntax can be replaced by a comma ',' as per a number of the above examples.

**NOTE 2**

On the THOR range (v4.02+) the word 'TO' in the prompts is replaced by the symbol =>

**NOTE 3**

On the THOR range, the prompt message are altered from 'Y/N/A/Q' to 'Yes/No/All/Quit'.

**NOTE 4**

As with COPY, WCOPY does not copy the header to serial devices (eg. ser) if this is specified as the destination. However, the THOR variant of this command actually looks to see whether the file-type or file dependent information fields are non-zero in which case the header is always copied.

**NOTE 5**

If you have level-2 device drivers, any sub-directories in the specified source directory are ignored by WCOPY. For example, if:

```
DIR flp1_
```

gave the following result:

```
Psion Disk
400/1440 sectors
QUILL->
ABACUS->
```

Then:

```
WCOPY flp1_
```

would have no effect. However, compare:

```
WCOPY flp1_QUILL_
```

which would copy all of the files in the sub-directory 'QUILL' to the current destination device.

**NOTE 6**

Both parameters must be supplied for the THOR variant of this command, otherwise the error 'Bad Parameter' will be reported.

**NOTE 7**

Current versions (at least up to v2.85) of WCOPY do not work correctly with the DEV device when this is pointing at a sub-directory (eg:

```
DEV_USE 1, flp1_QUILL_: WCOPY DEV1_
```

WCOPY will however work if the DEV device is pointing at a root directory, eg:

```
DEV_USE 1, flp1_.
```

**CROSS-REFERENCE**

*SPL_USE* and *DEST_USE* set the destination device. See *COPY*, *WCOPY_F* and *WCOPY_O* which are all similar. *WREN*, *WDIR*, *WSTAT* and *WDEL* all use wildcards. *COPY* and *SPL* allow you to copy specific files.

## 30.4 WCOPY_F

| | |
|---|---|
| Syntax | WCOPY_F [#ch,] wild1 TO wild2 |
| Location | THORs |

This command works in a similar way to WCOPY. However, although it lists the files being copied to the given channel (default #0), the user is not prompted to confirm that each file should be copied. The user will however be asked to confirm should the destination filename already exist.

**CROSS-REFERENCE**

See *WCOPY*.

## 30.5 WCOPY_O

| | |
|---|---|
| Syntax | WCOPY_O [#ch,] wild1 TO wild2 |
| Location | THORs |

WCOPY_O is the same as WCOPY_F except that any existing files are automatically overwritten without any prompting.

**CROSS-REFERENCE**

See *WCOPY_F*.

## 30.6 WDEL

| | |
|---|---|
| Syntax | WDEL [#ch,] [wild] |
| Location | Toolkit II, THORs |

WDEL allows you to delete several files which match the given wildcard at the same time. If wild contains a device name, then each file on that device is checked to see if its name matches the wildcard, otherwise the files on the default data directory are checked.

If any files are found which match the wildcard, a prompt will appear in the specified window (default #0) to the effect:

```
filename..Y/N/A/Q?
```

You must then either press <Y> to delete the offered file, <N> to leave that file, <A> to delete that file and all other files which match the wildcard, or <Q> to stop WDEL. <ESC> and <CTRL><SPACE> will have the same effect as <Q>.

**Example**

```
WDEL win1_v1_
```

will delete all files in the sub-directory v1.

**NOTE 1**

The THOR variant of WDEL has amended the prompt to read: 'Yes/No/All/Quit'

**NOTE 2**

Current versions of WDEL (at least up to v2.88) do not work with the DEV device when this is pointing to a sub-directory. Even if you can persuade WDEL to offer you the filename for deletion, when you press <Y> or <A>, WDEL fails to delete the file!

**NOTE 3**

If you try to use WDEL on a write protected disk, it will ask you whether you want to delete each file in turn reporting for each filename that the disk is write-protected, rather than stopping altogether.

**CROSS-REFERENCE**

*WCOPY* provides details about wildcards. *DELETE* allows you to delete single files.

## 30.7 WDEL_F

| | |
|---|---|
| Syntax | WDEL_F [#ch,] [wild] |
| Location | THORs |

WDEL_F is exactly the same as WDEL except no prompts or information about the files being deleted is shown on screen.

**CROSS-REFERENCE**

See *WDEL*.

## 30.8 WDIR

| | |
|---|---|
| Syntax | WDIR [#ch,] [wild] or<br>WDIR \file [,wild] (Toolkit II only) |
| Location | Toolkit II, THORs |

WDIR allows you to produce a list of all of the filenames on a given medium which match with the specified wildcard. If wild contains a device name, then a list of all of the files on that device which match with the wildcard is printed out to the specified channel (default #1). If however, a device is not specified, the default data device is used.

The second variant is only supported by Toolkit II and allows you to send the results to the specified file instead of sending it to a channel. If file does not include a valid device, the default data device is used, and if the file already exists, you will be asked whether or not you wish to overwrite it. The file is then opened by the WDIR command, the list of files written to it and then closed again.

**Examples**

```
WDIR \ser1, flp1___scr
```

will produce a list of all of the files on flp1_ whose names end with _scr.

```
WDIR my
```

lists all files in the current directory which start with my.

```
WDIR _my
```

lists files which start with my or contain _my somewhere.

**CROSS-REFERENCE**

*DIR* will produce a list of all of the files on a given medium. *WCOPY* contains details of how wildcards operate.

## 30.9 WEEKDAY%

| | |
|---|---|
| Syntax | WEEKDAY% [datestamp] |
| Location | SMSQ/E |

This function complements the *DATE* and *DATE$* functions, by returning the day of the week as a number starting from 0 for Sunday corresponding to the given datestamp, or current date, if no datestamp was given.

**Examples**

```
PRINT WEEKDAY% (0)
```

will print the month part of the QL's epoch, 0 for Sunday, January 1st, 1961

```
PRINT WEEKDAY%
```

will print the current weekday number, (0...6 for Sunday to Saturday).

**CROSS-REFERENCE**

See *DATE*, *YEAR%*, *MONTH%*, *DAY%*.

## 30.10 WGET

| | |
|---|---|
| Syntax | WGET [#ch\position,] [item *[,item$^i$]* ..] or |
| Syntax | WGET [#ch,] [item *[,item$^i$]* ..] |
| Location | SMSQ/E |

This command is very similar to BGET, except that this fetches a word (in the range 0..65535) from the given channel (default #3).

**CROSS-REFERENCE**

See *WPUT* and *BGET*.

## 30.11 WHEN condition

| | |
|---|---|
| Syntax | WHEN condition |
| Location | QL ROM (post JM), THOR XVI, Not SMSQ/E |

WHEN is used to identify the start of a SuperBASIC structure which is used to surround lines of SuperBASIC code which should be executed whenever the given condition is met. The condition is not checked when a variable is READ, or INPUT.

The syntax of the SuperBASIC structure can take two forms:

WHEN condition:statement:sup:*[:statement]*

or

WHEN condition *[statements]* .. END WHEN

The condition can be anything which is accepted by the IF command, provided that it begins with the name of a variable (for example, WHEN a-10=b is acceptable, but WHEN 10-a=b is not). The variable cannot be an array.

When a program is run, the interpreter will make a note of the variable being tested and then jump to the statement following the END WHEN statement (unless the in-line format is used when control jumps to

the next line if END WHEN does not appear on that line). Great care must however, be taken where the condition refers to more than one variable, as an 'error in expression' will be reported if a variable is not defined when the condition is tested, for example, the following stops with 'error in expression' at line 4:

```
4 WHEN x>1 AND y>1
5   x=x+1:PRINT 'hello'
6 END WHEN
7 PRINT 'Start'
8 :
100 FOR x=1 TO 2
110   FOR y=1 TO 2
120     PRINT x,y;' ';
130   END FOR y
140 END FOR x
```

This is because when line 100 is processed, the interpreter jumps to the WHEN clause. At this stage, y is undefined, hence the error. The program will work if you add the line:

```
1 y=0
```

Although blocks can be specified which check for various conditions of the same variable, if the conditions overlap, there is no guarantee as to which WHEN statement will be executed first. Blocks cannot be mixed together. In the following example, although if a$='me' the messages 'hello' and 'who' will be printed, and if a=2 the only message which will be printed is 'A is 2' - when the program is RUN, the first END WHEN command is matched with line 1, thus the message 'who' is also printed when the program is run (it is extremely bad programming practice in any event to mix program structures of this sort).

```
1 WHEN a$='me'
2   PRINT 'hello'
3   WHEN a=2
4     PRINT 'A is 2'
5   END WHEN
6   PRINT 'Who'
7 END WHEN
```

WHEN processing is turned off by the command WHEN anything, and also when the NEW, CLEAR, LOAD, LRUN, MERGE, and MRUN commands are issued. You can also switch off WHEN processing on a given variable (eg. b) by the command WHEN b (in the following example).

**Example**

```
110 WHEN a>100 AND a<1000: PRINT 'A is now in the range 100-1000': a=a+100
120 WHEN b=a
130   PRINT 'B is now the same as A ': PRINT B,A: A=A+50
140 END WHEN
150 WHEN b MOD 100=0: b=b+200
155 :
160 LET a=100: b=a
170 a=10
180 REPeat Loop
190   a=a+1: b=b-1
```

(continues on next page)

(continued from previous page)

```
200   AT 0,0: PRINT 'A='!a\\'B='!b
210 END REPeat Loop
```

**NOTE 1**

This command does not work reliably on any QL versions other than Minerva v1.77 or later: although Toolkit II improves the reliability, problems include calling the block more than once, and reporting 'bad name' when the block is called. WHEN clauses will also remain in force despite NEW, CLEAR, LRUN, LOAD, MERGE and MRUN, unless Toolkit II is present.

**NOTE 2**

A WHEN clause will not be called if it is already active, even though the program may have jumped out of the actual WHEN clause. For example:

```
100 WHEN a=100: PRINT 'A=100': GOTO 400
115 :
110 a=10
120 REPeat loop
130   a=a+10: PRINT a
140 END REPeat loop
150 STOP
160 :
400 FOR a=10 TO 200 STEP 30
410   PRINT a
420 END FOR a
```

**NOTE 3**

On JS MG and THOR XVI ROMs, a maximum of 20 WHEN clauses can be active at any time.

**CROSS-REFERENCE**

Other SuperBASIC structures are *WHEN ERRor*, *SELect ON* and *IF*..*END IF*.

*END WHEN* defines the end of a WHEN XXX structure.

## 30.12 WHEN ERRor

| | |
|---|---|
| Syntax | WHEN ERRor |
| Location | QL ROM (post JM), THOR XVI |

This command marks the beginning of the SuperBASIC structure which is used to surround lines of SuperBASIC code which should be executed whenever an error is generated whilst error trapping is active. Error trapping is activated as soon as the interpreter reads a line containing WHEN ERRor. It is therefore not activated by a WHEN ERRor command being entered into the command window (#0) - indeed this has a special purpose (see below). The syntax of the SuperBASIC structure can take two forms:

WHEN ERROR: statement *[:statement]*

or

WHEN ERROR *[statements]* .. END WHEN

In the normal course of progress, the WHEN ERRor block would appear at the start of a SuperBASIC program, and error trapping would therefore be enabled as soon as a program is RUN. Once error trapping is enabled, whenever an error is generated, control is passed to the WHEN ERRor clause, allowing you to specify how it the error to be dealt with.

It must however be borne in mind that whilst active, errors will trigger the WHEN ERRor clause whether they are generated whilst the program is being RUN or at some other stage (eg. if a direct command causes an error). If the interpreter comes across more than one WHEN ERRor block, then the latest one is used to trap errors.

Errors generated within the WHEN ERRor block itself are reported as normal, although the message 'during WHEN processing' is displayed along with the error message. Unless you include a STOP statement in the WHEN ERRor clause, after going through all of the lines within the clause, the program will continue running from the statement following the one which caused the error.

You can force this to happen with CONTINUE, whereas RETRY can be used to re-execute the command which caused the error. Error trapping is turned off by the command WHEN ERRor (when entered as a direct command), and also when the NEW, CLEAR, LOAD, LRUN, MERGE, and MRUN commands are issued.

**Example**

A program which provides a fully error trapped educational aid:

```
100 WHEN ERRor
110   STRIP#0,2
120   IF ERR_XP
130     PRINT#0,'Please enter a number!'\'Press a key'
140     PAUSE:STRIP #0,0:RETRY 320
150   END IF
160   IF ERR_OV
170     PRINT#0,'Divide by zero is undefined!'\'Press a key'
180     PAUSE:STRIP #0,0:RETRY 320
190   END IF
200   STRIP #0,0
210   PRINT #0,'At line: ';ERLIN:REPORT:STOP
220 END WHEN
225 :
230 MODE 8
240 WINDOW 448,200,32,16:PAPER 0:INK 6:CLS
250 WINDOW #0,448,40,32,216:PAPER#0,0:INK#0,7:CLS#0
260 CSIZE 2,0:AT 8,8:PRINT 'Maths Division Tutor'
270 CSIZE 1,0
280 REPeat loop
290   y=RND(1 TO 10):x=RND(1 TO 10)*y
300   IF y>x:ya=x:x=y:y=ya
310   IF RND>.9:x=0:y=0
320   REPeat answer
330     AT 10,0:CLS 2:AT 11,0:CLS#0
340     INPUT 'Enter number to divide'!(x)!'by to give'!(y)!': ';a
350     IF x/a=y THEN EXIT answer
360     PRINT \\'Wrong - Please try again'\'Press a key'
370     PAUSE
```

(continues on next page)

(continued from previous page)

```
380   END REPeat answer
390   PRINT \\'Correct - Another one...'\'Press a key'
400   PAUSE
410 END REPeat loop
```

**NOTE 1**

This SuperBASIC structure does not work very reliably on any QL versions other than Minerva v1.77 (or later), SMS or the THOR XVI: although Toolkit II improves the reliability, problems include crashing the machine if an error is generated inside a function whilst error trapping is enabled {eg. PRINT SQRT(-1)}, or if you try to carry out INKEY$ at the end of a file. WHEN ERRor clauses will also remain in force despite NEW, CLEAR, LRUN, LOAD, MERGE and MRUN.

**NOTE 2**

WHEN ERRor cannot trap the Break key <CTRL><SPACE> (and <ESC> on Minerva), which will continue to stop a SuperBASIC program.

**NOTE 3**

You should not try to nest several WHEN ERRor clauses - under SMS the error 'WHEN clauses may not be nested' is reported.

**SMS NOTE**

Even in the in-line version of WHEN ERRor it is imperative that END WHEN is specified, otherwise the error 'Incomplete WHEN clause' will be reported.

**CROSS-REFERENCE**

*ERLIN* returns the line number on which the error occurred. *ERNUM* returns the error number itself. There are several functions in the form *ERR_XX* which return 1 if the given error has occurred. *BREAK_OFF* allows you to turn the Break key off. *END WHEN* defines the end of the error handling block.

## 30.13 WHERE_FONTS

| | |
|---|---|
| Syntax | address = WHERE_FONTS(#channel, 1_or_2) |
| Location | DJToolkit 1.16 |

This function returns a value that corresponds to the address of the fonts in use on the specified channel. The second parameter must be 1 for the first font address or 2 for the second, there are two fonts used on each channel. If the result is negative then it will be a normal QDOS error code. The channel must be a CON_ or a SCR_ channel to avoid errors.

**EXAMPLE**

The following example will report on the two fonts used in any given channel, and will display the character set defined in that font:

```
4480 DEFine PROCedure REPORT_ON_FONTS (channel)
4485   LOCal address, lowest, number, b
4490   REMark show details of channel's fonts
```

(continues on next page)

(continued from previous page)

```
4495   CLS
4500   FOR a = 1,2
4505     address = WHERE_FONTS(#channel, a)
4510     lowest = PEEK(address)
4515     number = PEEK(address + 1)
4520     PRINT '#'; channel; ' font '; a; ' at address '; address
4525     PRINT 'Lowest character code = '; lowest
4530     PRINT 'Number of characters  = '; number + 1
4535     REMark print all but default characters
4540     PRINT : REMark blank line
4545     FOR b = lowest + 1 TO lowest + number :PRINT CHR$(b);
4550     PRINT \\\ : REMark 2 blank lines
4555   END FOR a
4560 END DEFine REPORT_ON_FONTS
```

## 30.14 WIDTH

| | |
|---|---|
| Syntax | WIDTH [#channel,] x |
| Location | QL ROM |

The WIDTH command is an output formatting command which allows the user to specify the width of a device which is being used by the QL for output (such as a printer) on the given channel (default #1). This can only be used on non-screen (ie. not scr_ or con_) channels and only has any effect if you use one of the separators exclamation mark (!); comma (,) or TO when PRINTing.

The value of x should represent the number of characters wide which the output device is to use (the default is 80 characters).

**Example**

A short procedure to output text to a non-screen device of a given width without chopping off any words at the end of each line:

```
100 :
110 t$ = 'The way in which the WIDTH command works is very particular to the␣
↪QL '
120 t$ = t$ & 'and is really only suited for specific types of work. If you␣
↪do not '
130 t$ = t$ & 'use the separators ! or , then the text will still be output␣
↪at the '
140 t$ = t$ & 'default width of 80'
115 :
200 OPEN_NEW #3,ram2_junk
210 DUMP_TEXT #3, t$, 80
220 DUMP_TEXT #3, t$, 40
230 :
240 CLOSE#3
250 :
260 :
```

(continues on next page)

(continued from previous page)

```
1000 DEFine PROCedure DUMP_TEXT(chan,str$,wid)
1010   LOCal word$
1020   WIDTH #chan,wid
1030   IF str$="" THEN RETurn
1040   word_start=1
1050   REPeat word_loop
1060     word_end=(' ' INSTR str$)-1
1070     IF word_end>=word_start
1080       word$=str$(word_start TO word_end)
1090     ELSE
1100       word$=str$(word_start TO )
1110     END IF
1120     PRINT #chan;!word$!:PRINT !word$!:PAUSE
1130     IF word_end+2>LEN(str$) OR word_end=-1:EXIT word_loop
1140     str$=str$(word_end+2 TO )
1150   END REPeat word_loop
1160 END DEFine
```

**CROSS-REFERENCE**

See *OPEN* and *PRINT*.

## 30.15 WINDOW

| | |
|---|---|
| Syntax | WINDOW [#ch,] x, y, posx, posy or<br>WINDOW [#ch,] x, y, posx, posy [\border] (Minerva v1.79+, THOR XVI) |
| Location | QL ROM, Minerva, THOR XVI |

This command redefines the given screen window (default #1) by specifying the new size and position of the window. The values must all be calculated in the pixel co-ordinate system, which means that x and posx can be in the range 0…XLIM (in both MODE 4 and MODE 8), provided that x+posx<=XLIM and y and posy can be in the range 0..YLIM, provided that y+posy<=YLIM.

On a standard QL resolution screen (ie. 512x256 pixels), due to the shape of the screen, a window which measures 100x100 pixels will not appear square. You will need to use a size of 137x100 pixels instead! The Minerva and THOR XVI variants allow you to specify a border to be drawn around the window at the same time, by the addition of up to a further four parameters in the form: [\border_size [,colour [,colour2 [,stipple ]]]] This therefore allows you to combine the WINDOW and BORDER commands. For example:

```
WINDOW 448,200,32,16\2,2
```

is the same as:

```
WINDOW 448,200,32,16:cBORDER 2,2.
```

**Example**

```
WINDOW 448,200,32,16
```

is similar to:

```
OPEN #1,CON
```

**NOTE 1**

Although the '\' separator is not checked for on the Minerva and THOR XVI implementations, it is recommended to ensure that this is present to ensure future compatibility. Older ROM versions did not check the number of parameters, which could result in some software causing problems unless the separator is actually checked for.

**NOTE 2**

You cannot have a gap of one pixel between windows, even in MODE 4 - this is to ensure compatibility between MODE 4 and MODE 8. Any odd parameters will be rounded down.

**MINERVA NOTE**

In a MultiBasic, both channel #0 and #1 are inextricably linked. Unfortunately, this means that in certain cases both channel #0 and channel #1 must have the same size and position: any attempt to re-size #0 will re-size #1 and vice versa. See the MultiBasic appendix for further details.

**CROSS-REFERENCE**

*OPEN* allows you to open a window ready for use. *BORDER* allows you to set an implicit border.

## 30.16 WINF$

| | |
|---|---|
| Syntax | WINF$ |
| Location | Fn |

This is the same as *WMAN$*.

## 30.17 WIN2

| | |
|---|---|
| Syntax | WIN2 directory |
| Location | Gold Card, THOR XVI and ST/QL (Level C-19+) |

This command simulates the drive win2_ if only one hard disk (win1_) is present. All access to win2_ will be redirected to directory.

**Example**

WIN2 system: DIR *win2*

will produce a listing of the files held in the sub-directory win1_system. This is equivalent to:

```
DIR win1_system
```

**NOTE**

Do not specify the device as part of directory.

**CROSS-REFERENCE**

*DEV_USE* is much more flexible.

## 30.18 WIN_BASE

| | |
|---|---|
| Syntax | WIN_BASE [(#channel)] |
| Location | Fn |

This function returns the start address of the definition block for the specified window (default #1). If an error occurs WIN_BASE returns the appropriate QDOS error code, eg. -15 if the channel does not apply to a window or -6 if the channel is not open.

**Example**

Some information about the internal structure of QDOS is necessary to make use of WIN_BASE from SuperBASIC. This function returns the PAPER background colour of a window:

```
100 DEFine FuNction GET_PAPER (winchan)
110   IF WIN_BASE(#winchan)<0 THEN
120     PRINT#0,"GET_PAPER: ";: REPORT #0, WIN_BASE(#winchan)
130     PAUSE 800: STOP
140   END IF
150   RETurn PEEK(WIN_BASE(#winchan)+68)
160 END DEFine GET_PAPER
```

**NOTE**

The Window Manager changes the structure of window definition blocks.

**CROSS-REFERENCE**

*SYS_BASE*, *SET*

## 30.19 WIN_CHECK

| | |
|---|---|
| Syntax | WIN_CHECK drive |
| Location | SMSQ/E for Q68 |

Checks whether a WIN container file on the card is indeed in contiguous sectors on the card. If the command does not return an error, the container file corresponding to the drive is OK.

## 30.20 WIN_DRIVE

| | |
|---|---|
| Syntax | WIN_DRIVE driveno [, unit, disk] or<br>WIN_DRIVE driveno, unit [,disk] [,partition ](SMSQ/E only) or<br>WIN_DRIVE driveno, path$(QPC & QXL SMSQ/E only) or<br>WIN_DRIVE driveno, card, path$(Q68 SMSQ/E only) |
| Location | ST/QL, SMSQ/E for Atari, Q68, and QXL / QPC |

It is possible not only to have several hard disk units attached to the Atari ST, but each hard disk unit can also have more than one drive in it (for example, you might own a hard disk unit which has both a standard hard disk and a changeable hard disk inside).

The normal chain of events is that each WIN drive would attach itself to the equivalent hard disk unit, for example, WIN1_ would be connected to hard disk unit 0, WIN2_ to hard disk unit 1 and so on. However, so that you may link the WIN drives to specific disks within each unit, the WIN_DRIVE command exists.

WIN_DRIVE takes the WIN drive number supplied by driveno and will attach this to the specified disk which is housed in the specified unit.

Driveno must be in the range 1…8 - this corresponds to the number which will be attached to WIN to refer to the relevant drive (eg. WIN4_). If a unit and disk are not specified, this command will remove the definition attached to the specified driveno.

Unit should be in the range 0…7 and represents the number of the disk drive controller. An internal disk drive controller is normally unit 0, but external controller unit numbers will depend upon the setting of the switches on the back of the box.

Card should be in the range 1…2 and represents the number of the SD card reader containing the file.

If you are running SMSQ/E on the TT and wish to access a SCSI disk controller, then you will need to add 8 to the value of unit.

Disk can be in the range 0…7 and represents the number of the disk drive actually addressed by the given controller. It is however rare in the Atari world to have more than one disk drive per controller and so this value is normally either 0 or 1. The default is 0.

Finally, each disk can be partitioned, so that an area of each disk is set aside for specific uses (eg. for QDOS or for GEM). You therefore need to specify the number of the partition. Default is 0. Although you can configure SMSQ/E to start from a specific drive and partition, it normally looks for a BOOT file in any partition on unit 0 (on the TT it will look at SCSI unit 0 and then ASCI unit 0). If found, WIN1_ will be set to this partition.

In current versions of SMSQ/E WIN2_ will not be linked to anything until you use the WIN_DRIVE command.

**Example**

Assume that you have two hard disk units plugged into the Atari ST, the first one of which (unit 0) contains a normal hard disk unit (disk 0) and a changeable hard disk unit (disk 1).

On starting the Emulator, WIN1_ would refer to the normal hard disk in unit 0 and WIN2_ would be undefined. You could not therefore access the changeable hard disk from the Emulator. To avoid this, use the commands:

```
WIN_DRIVE 2,0,1,0
WIN_DRIVE 3,1,0,2
```

This will link WIN2_ to the changeable hard disk (this is disk number 1 in unit 0, partition 0) and WIN3_ would then point to the hard disk in the second unit (disk 0 in unit 1, partition 2).

**NOTE**

Disk must be specified unless it is 0. - this means that if three parameters are specified, the third parameter is taken to be the partition number.

**QPC / QXL NOTE**

From v2.89 of SMSQ/E, WIN_DRIVE is implemented slightly differently on these emulators. For each driveno, you can specify a PC related path for the hard disk (the hard disk under QPC and QXL is implemented as a single file stored on the PC's hard disks). For example, use:

```
WIN_DRIVE 2,'D:\qxl.win'
```

to make win2_ on the QL emulator look use the file qxl.win on the PC's D: drive. In this way, CD-ROMs and DVD-RAMs can be used on the PC as a hard drive for the QL emulator. Although QPC allows you to have several QL hard disk files on each PC device, QXL only allows one qxl.win file per PC device!!

**WARNING 1**

You must not make the QDOS WIN drive point to another physical drive if that WIN device has been accessed already. For example, if you wanted to follow the above example, but had just loaded a program from WIN2_ you *must not* use:

```
WIN_DRIVE 2,0,1.
```

**WARNING 2**

Do not attempt to make two WIN drives point to the same physical drive!

**CROSS-REFERENCE**

*WIN_DRIVE$* returns the parameters already associated with a WIN drive. *WIN_FORMAT* allows you to format a hard disk.

## 30.21 WIN_DRIVE$

| | |
|---|---|
| Syntax | WIN_DRIVE$ (drive) |
| Location | SMSQ/E for Atari, Q68, and QXL / QPC |

On SMSQ/E for the Atari, this function returns a string containing the unit, disk and partition numbers addressed by the specified WIN drive.

Under SMSQ/E for the QXL and QPC (v2.89+), this function will return a string indicating the file on the PC which is used as that hard drive.

If the specified drive has not been linked to any particular hard disk partition, an empty string is returned.

**Atari Examples**

```
WIN_DRIVE 2,0,1,0
PRINT WIN_DRIVE$(2): REMark Will print 0,1,0
```

**Q68 Examples**

```
WIN_DRIVE 2,1,"drive.win"
PRINT WIN_DRITE$(2): REMark will print DRIVE.WIN,1
```

**QXL / QPC Examples**

```
WIN_DRIVE 2,'C:\qxlback.win'
PRINT WIN_DRIVE$(2): REMark will print C:\qxlback.win
```

**CROSS-REFERENCE**

See *WIN_DRIVE*.

## 30.22 WIN_FORMAT

| | |
|---|---|
| Syntax | WIN_FORMAT drive [,protect] |
| Location | SMSQ/E (v2.73+) for Atari, Q68, and QXL / QPC |

In order to prevent you from accidentally formatting your hard disk (or a partition of your hard disk) and overwriting important information, SMSQ/E has implemented a form of protection. Before formatting a QDOS partition, you will first of all need to create that partition using either the Atari's or the PC's operating system (see the SMSQ/E documentation for details). You must then use the WIN_DRIVE command, followed by WIN_FORMAT to allow the FORMAT command to work on the hard disk.

Protect is a flag - if it is omitted, this removes the protection from the partition pointed to by the specified WIN drive. protect=1 sets the protection again after FORMATting.

**Example**

To format a QDOS partition called PROGS, pointed to by WIN2 on unit 1, partition 1:

```
WIN_DRIVE 2,1,1
WIN_FORMAT 2
FORMAT win2_PROGS
WIN_FORMAT 2,1
```

**NOTE**

Earlier versions of SMSQ/E did not include this command and the FORMAT command would work once WIN_DRIVE had been used to set up the WIN drive name.

**CROSS-REFERENCE**

See *FORMAT* and *WIN_DRIVE*.

## 30.23 WIN_REMV

| | |
|---|---|
| Syntax | WIN_REMV driveno, flag (SMSQ/E & ST/QL Level C-24+) or<br>WIN_REMV driveno |
| Location | ST/QL (Level C-20+), SMSQ/E for Atari, QXL / QPC |

The advent of changeable hard disk drives caused a lot of problems, since it is just about feasible that you might try to remove the hard disk unit whilst it is being accessed, which can cause serious damage to the drive unit. Although the drives attempt to warn the computer when they are and are not removable, it is next to impossible to ensure that when the drive says it can be removed, it is not actually powering up or down.

The command WIN_REMV tells the system that the drive connected to the specified port is a removable hard disk drive - the door on the unit will then remain firmly locked as long as any files on the hard disk are open.

Note that driveno must be in the range 1…8. SMSQ/E allows the first variant - flag can be omitted which is equivalent to 1 (signifies a removable hard disk). It can also be one of the following values:

- 0: Clear the removable flag from the drive
- V: Mark the drive as being a VORTEX drive

**Example**

```
WIN_REMV 2
```

denotes win2_ as a removable disk drive.

**NOTE**

It is essential that WIN_REMV is used as early as possible - either before the drive is first accessed or as the first line of your boot program if the Emulator is being booted from the hard disk in question.

**SMSQ/E NOTE**

SMSQ/E manages to detect removable hard disks 100% on SCSI ports. It is also normally successful in detecting removable hard disks connected to ASCI ports unless you configure it to ignore them, therefore this command is only really needed on ASCI drives.

**QPC NOTE**

You need v1.43+ of QPC to use removable drives.

**WARNING**

Never try to remove a hard disk (removable or otherwise!) whilst it is running.

**CROSS-REFERENCE**

*WIN_STOP* will park the head on the drive prior to removal. *DMEDIUM_REMOVE* can tell you if the given device is a removable hard disk.

## 30.24 WIN_SAFE

| | |
|---|---|
| Syntax | WIN_SAFE card |
| Location | SMSQ/E for Q68 |

This command checks whether it is safe to remove a card as far as the WIN driver is concerned. If the card is safe to remove it returns no error. If the card is not safe to remove it returns the error "is in use".

**Example**

```
WIN_SAFE 1 REMark is SD card 1 safe to remove
```

## 30.25 WIN_SLUG

| | |
|---|---|
| Syntax | WIN_SLUG x |
| Location | ST/QL, SMSQ/E for Atari |

Some winchester (hard disk) ASCI drives, in particular the Megafile and Vortex drives, need a special parameter to be passed to them before they can be accessed by the QL due to timing faults in their controllers. WIN_SLUG allows you to set this parameter.

The value of x will depend upon the drive being used, and can be anything in the range 0…255. It is measured in units which are 0.8ms. This parameter sets the minimum time that must elapse between operations on the ASCI bus. Most controllers work with the default setting of 30 (which equates to a time of 2.5ms). Refer to the disk documentation for further details.

## 30.26 WIN_START

| | |
|---|---|
| Syntax | WIN_START driveno |
| Location | ST/QL, SMSQ/E for Atari and QPC / QXL |

After the head on a changeable hard disk drive has been parked, it is necessary to tell it to release its head before you can access the drive. WIN_START issues the command to do this. The parameter driveno is the number of the hard disk to be told to release the head. Driveno must be in the range 1…8.

**Example**

```
WIN_START 1
```

releases the head on win1_.

**NOTE**

Some hard disk drives will not release the head even after WIN_START unless the power to the drive is switched off and back on.

**CROSS-REFERENCE**

See also *WIN_DRIVE* and *WIN_STOP*.

## 30.27 WIN_STOP

| | |
|---|---|
| Syntax | WIN_STOP driveno |
| Location | ST/QL, SMSQ/E for Atari and QPC / QXL |

If you are going to move a computer around, or swap over a changeable hard disk drive, it is *essential* that you make sure that the head on the hard disk drive is parked. This basically means that the drive locks the head away and ensures that it cannot be banged onto the surface of the hard disk drive.

Some hard disk interfaces (such as the Miracle Hard Disk system for the QL) automatically park the head if the drive has not been accessed for a while. However, on other systems, it is necessary to do this explicitly. WIN_STOP tells the hard disk in the specified drive to park its head. driveno must be in the range 1…8.

**Example**

```
WIN_STOP 2
```

will park the head in win2_.

**WARNING 1**

Never move a hard disk about unless its head is parked as this can cause permanent damage to the drive.

**WARNING 2**

Some hard disk drives require that you park the head before disconnecting the power to the drive. Refer to the instructions for the hard disk which you are using.

**WARNING 3**

You may find that some drives will refuse to respond to access calls if stopped accidentally, or when using this command. If WIN_START does not revive them, then unfortunately the only thing to do is to reset the system (switching the power back and back on).

**CROSS-REFERENCE**

*WIN_START* releases the head so that the drive can be used again.

## 30.28 WIN_USE

| | |
|---|---|
| Syntax | WIN_USE [device] |
| Location | THOR XVI, ST/QL, Hard disk driver,<br>SMSQ/E for Atari, Q68, and QXL / QPC |

As with FLP_USE this allows you to assign another three letter description to the WIN device driver, so that it can be accessed by programs which do not allow you to alter their devices. If no device is specified, then the device name is returned to the default win.

**Example**

```
WIN_USE mdv
```

will ensure that any further attempt to access mdv1_ will actually access win1_. If you later use the command:

```
WIN_USE
```

or:

```
WIN_USE win
```

then you will once again be able to use the microdrives as well as win1_.

**NOTE**

The QL's operating system tests for directory device drivers in a fixed order: DEV, FLP, RAM, WIN and MDV. This means that if you rename a driver to three letters which refer to a device driver earlier in the list, that original device driver will be used in preference. For example:

```
WIN_USE flp
```

will not work (attempts to read a file from flp1_ will still try to read floppy disk drive number one) - you will need to also rename the floppy disk driver:

```
FLP_USE flp
```

**CROSS-REFERENCE**

*FLP_USE*, *RAM_USE*, *DEV_USE* are similar. *DMEDIUM_TYPE* can be used to find out the type of device which a name actually refers to. *DMEDIUM_NAME$* will return the default name of a device.

## 30.29 WIN_WP

| | |
|---|---|
| Syntax | WIN_WP drive, protect |
| Location | SMSQ/E for Atari, Q68, and QXL / QPC |

This command allows you to mark a specified WIN drive as read only protect=1 will write protect the hard disk. protect=0 (the default) will remove the write protection.

**CROSS-REFERENCE**

*DMEDIUM_RDONLY* will tell you if a device is read only. See also *WIN_REMV* and *WIN_FORMAT* for other types of protection.

## 30.30 WIPE

| | |
|---|---|
| Syntax | WIPE |
| Location | BeuleTools, WIPE |

This command clears the whole screen so that it is completely black.

WIPE is an alternative to:

```
OPEN#11,scr_512x256a0x0:
CLS#11:
CLOSE#11
```

or:

```
SCRBASE SCREEN: SCLR 0
```

**NOTE**

This command presumes that the screen starts at 131072 and measures 512x256 - it will therefore not work on higher resolutions.

**CROSS-REFERENCE**

*CLS* clears a window in its current paper colour, *SCLR* the (background) screen in a given colour. *CLS_A* is a global *CLS*.

## 30.31 WLD

| | |
|---|---|
| Syntax | WLD (word1$, word2$ [,dummy] ) or<br>WLD (word1$, word2$, w1, w2, w3 [,dummy] ) |
| Location | Ähnlichkeiten |

This function calculates the weighted Levenstein phonetic distance between two strings: the smaller the result, the more that the two strings are phonetically similar.

If two strings are found to be identical, then 0 is returned, otherwise a positive integer is returned.

The value of the dummy parameter does not actually matter - if it is present then the function will not distinguish between upper and lower case characters.

The three additional parameters of the second syntax allow you to alter the importance of three possible factors used to calculate the difference between the strings - each parameter should have a positive value:

- w1: wrong letters
- w2: strings too short
- W3: strings too long

**Example**

```
100 a$="Sinclair QL": b$="IBM PC": CLS
110 PRINT a$;" <-> ";b$
120 PRINT\WLD(a$,b$), WLD(a$,b$,0)
130 PRINT WLD(a$,b$,1,1,1), WLD(a$,b$,1,1,1,0)
140 PRINT WLD(a$,b$,0,0,0)
150 PRINT WLD(a$,b$,1,2,3), WLD(a$,b$,3,2,1)
```

**CROSS-REFERENCE**

*SOUNDEX*, *PHONEM*.

## 30.32 WM

| | |
|---|---|
| Syntax | WM |
| Location | WM |

QPAC2 uses a Button Frame which is normally situated across the top of the screen. The command WM sets up the three basic windows #0, #1 and #2 so that there is space for two rows of buttons. At the same time, the window attributes are reset to the status they would have been in had you reset the system and pressed <F1> for monitor mode. The current screen resolution mode is not affected.

**NOTE**

QPAC2 and the Pointer Environment are not necessary to use WM.

**CROSS-REFERENCE**

*WMON* restores the original monitor windows and *WTV* the TV mode. Use *INK*, *PAPER*, *BORDER* and *STRIP* to change window attributes.

## 30.33 WM_BLOCK

| | |
|---|---|
| Syntax | WM_BLOCK [#channel,] width, height, x, y, palette_index |
| Location | SMSQ/E >= 3.00 |

Newer Window Managers maintain a table of colour settings for programs to use as "standard colours". This is called the *System Palette*, also known as a 'colour theme'. Four system palette tables, or themes, are supplied with the operating system.

The list is sorted by *usage* rather than *colour* and includes colour values to be used for display items such as window background, border, loose items and so on. The items are referenced by a 4-digit hex number (16-bit value) as per the list under the entry for *WM_INK*, or the decimal number equivalent. These numbers should not be used in standard *INK*, *PAPER* and *BORDER* statements – they are not colour values, merely an index to an entry in a list of colour values. They should be used with the WM_x equivalent commands, which will look up the colour values to be used for the item numbers in the list.

WM_BLOCK draws a block in the channel indicated using the colour for the specified item number from the system palette.

**Example**

```
WM_BLOCK #1,100, 40, 0, 0, $201
```

Draws a block 100 pixels wide and 40 pixels high to #1 in the current system palette's window background colour.

**CROSS-REFERENCE**

See *WM_INK*, *WM_PAPER*, *WM_BORDER*, *WM_STRIP*.

## 30.34 WM_BORDER

| | |
|---|---|
| Syntax | WM_BORDER [#channel,] palette_index |
| Location | SMSQ/E >= 3.00 |

Newer Window Managers maintain a table of colour settings for programs to use as "standard colours". This is called the *System Palette*, also known as a 'colour theme'. Four system palette tables, or themes, are supplied with the operating system.

The list is sorted by *usage* rather than *colour* and includes colour values to be used for display items such as window background, border, loose items and so on. The items are referenced by a 4-digit hex number (16-bit value) as per the list under the entry for *WM_INK*, or the decimal number equivalent. These numbers should not be used in standard *INK*, *PAPER* and *BORDER* statements – they are not colour values, merely an index to an entry in a list of colour values. They should be used with the WM_x equivalent commands, which will look up the colour values to be used for the item numbers in the list.

WM_BORDER sets the border colour for the channel indicated to the colour for the specified item number from the system palette.

**Example**

```
WM_BORDER #1,$20e
```

Sets the border colour in #1 to the information window border colour from the current system palette.

**CROSS-REFERENCE**

See *WM_INK*, *WM_PAPER*, *WM_STRIP*, *WM_BLOCK*.

## 30.35 WM_INK

| | |
|---|---|
| Syntax | WM_INK [#channel,] palette_index |
| Location | SMSQ/E >= 3.00 |

Newer Window Managers maintain a table of colour settings for programs to use as "standard colours". This is called the *System Palette*, also known as a 'colour theme'. Four system palette tables, or themes, are supplied with the operating system.

The list is sorted by *usage* rather than *colour* and includes colour values to be used for display items such as window background, border, loose items and so on. The items are referenced by a 4-digit hex number (16-bit value) as per the list below, or the decimal number equivalent. These numbers should not be used in standard *INK*, *PAPER* and *BORDER* statements – they are not colour values, merely an index to an entry in a list of colour values. They should be used with the WM_x equivalent commands, which will look up the colour values to be used for the item numbers in the list.

WMINK sets the ink colour for the channel indicated to the colour for the specified item number from the system palette.

| Number | Meaning |
|---|---|
| $0200 | Window border |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| $0201 | Window background |
| $0202 | Window foreground |
| $0203 | Window middleground |
| $0204 | Title background |
| $0205 | Title text background |
| $0206 | Title foreground |
| $0207 | Loose item highlight |
| $0208 | Loose item available background |
| $0209 | Loose item available foreground |
| $020a | Loose item selected background |
| $020b | Loose item selected foreground |
| $020c | Loose item unavailable background |
| $020d | Loose item unavailable foreground |
| $020e | Information window border |
| $020f | Information window background |
| $0210 | Information window foreground |
| $0211 | Information window middleground |
| $0212 | Subsidiary information window border |
| $0213 | Subsidiary information window background |
| $0214 | Subsidiary information window foreground |
| $0215 | Subsidiary information window middleground |
| $0216 | Application window border |
| $0217 | Application window background |
| $0218 | Application window foreground |
| $0219 | Application window middleground |
| $021a | Application window item highlight |
| $021b | Application window item available background |
| $021c | Application window item available foreground |
| $021d | Application window item selected background |
| $021e | Application window item selected foreground |
| $021f | Application window item unavailable background |
| $0220 | Application window item unavailable foreground |
| $0221 | Pan/scroll bar |
| $0222 | Pan/scroll bar section |
| $0223 | Pan/scroll bar arrow |
| $0224 | Button highlight |
| $0225 | Button border |
| $0226 | Button background |
| $0227 | Button foreground |
| $0228 | Hint border |
| $0229 | Hint background |
| $022a | Hint foreground |
| $022b | Hint middleground |
| $022c | Error message background |
| $022d | Error message foreground |
| $022e | Error message middleground |
| $022f | Shaded area |
| $0230 | Dark 3D border shade |
| $0231 | Light 3D border shade |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| $0232 | Vertical area fill |
| $0233 | Subtitle background |
| $0234 | Subtitle text background |
| $0235 | Subtitle foreground |
| $0236 | Menu index background |
| $0237 | Menu index foreground |
| $0238 | Separator lines etc. |

**Example**

```
WM_INK #1,$206
```

Sets the foreground colour in #1 to the title window foreground.

**CROSS-REFERENCE**

See *WM_PAPER*, *WM_STRIP*, *WM_BORDER*, *WM_BLOCK*.

## 30.36 WM_MOVEMODE

| | |
|---|---|
| Syntax | WM_MOVEMODE mode |
| Location | SMSQ/E >= 3.01 |

Sets the mode in which windows are moved.

Modern window managers allow moving a window about the screen in various ways:

0 - the "classic" way - the pointer changes to the "move window" sprite which is moved about the screen.

1 - "Outline": click on the move icon with the MOUSE - keep holding the button down -, an outline of the window will appear which you can move around and position where you want it. Release the mouse button and the window positions itself correctly.

2 - "Full window". This is the same as 1 above, but instead of an outline, the entire window contents will be displayed during the movement. For Q40/Q60 users, switching on the Cache is advisable…

3 - "Full window with transparency" (implemented in SMSQ/E v. 3.16). This is the same as 2 above, but the window to be moved is made "transparent" : one can "see through" it. This is done via "alpha blending". Alpha blending requires a lot of computing power. So, even if your machine can theoretically handle this type of move, in practice it might not be feasible. For Q40/Q60 users, switching on the Cache is advisable.

**Example**

```
WM_MOVEMODE 0
```

Sets the window move mode to "classic", i.e. moving with the move icon.

**NOTE 1** In any but move mode 0 windows cannot be moved by the keyboard and strictly require a mouse. When moving windows with the keyboard, the move falls back to the "classic" icon move for this operation.

**NOTE 2** "Move with transparency" (mode 3) is only implemented for display modes where alpha blending actually makes sense, i.e. modes 16, 32 and 33. In other display modes, such as the QL screen modes, or Atari mono modes, this will be redirected to move mode 2.

**NOTE 3** The move modes are configured on a system-wide basis - you cannot have one job moving in mode 0 and the other in mode 1.

**NOTE 4** The window move mode can be configured in the operating system config blocks.

## 30.37 WM_PAPER

| | |
|---|---|
| Syntax | WM_PAPER [#channel,] palette_index |
| Location | SMSQ/E >= 3.00 |

Newer Window Managers maintain a table of colour settings for programs to use as "standard colours". This is called the *System Palette*, also known as a 'colour theme'. Four system palette tables, or themes, are supplied with the operating system.

The list is sorted by *usage* rather than *colour* and includes colour values to be used for display items such as window background, border, loose items and so on. The items are referenced by a 4-digit hex number (16-bit value) as per the list under the entry for *WM_INK*, or the decimal number equivalent. These numbers should not be used in standard *INK*, *PAPER* and *BORDER* statements – they are not colour values, merely an index to an entry in a list of colour values. They should be used with the WM_x equivalent commands, which will look up the colour values to be used for the item numbers in the list.

WM_PAPER sets the paper colour for the channel indicated to the colour for the specified item number from the system palette.

**Example**

```
WM_PAPER #1,$204
```

Sets the paper colour in #1 to the title window background colour from the current system palette.

**CROSS-REFERENCE**

See *WM_INK*, *WM_STRIP*, *WM_BORDER*, *WM_BLOCK*.

## 30.38 WM_STRIP

| | |
|---|---|
| Syntax | WM_STRIP [#channel,] palette_index |
| Location | SMSQ/E >= 3.00 |

Newer Window Managers maintain a table of colour settings for programs to use as "standard colours". This is called the *System Palette*, also known as a 'colour theme'. Four system palette tables, or themes, are supplied with the operating system.

The list is sorted by *usage* rather than *colour* and includes colour values to be used for display items such as window background, border, loose items and so on. The items are referenced by a 4-digit hex number (16-bit value) as per the list under the entry for *WM_INK*, or the decimal number equivalent. These numbers should not be used in standard *INK*, *PAPER* and *BORDER* statements – they are not

colour values, merely an index to an entry in a list of colour values. They should be used with the WM_x equivalent commands, which will look up the colour values to be used for the item numbers in the list.

WM_STRIP sets the strip colour for the channel indicated to the colour for the specified item number from the system palette.

**Example**

```
WM_STRIP #1,$204
```

Sets the strip colour in #1 to the title window background colour from the current system palette.

**CROSS-REFERENCE**

See *WM_INK*, *WM_PAPER*, *WM_BORDER*, *WM_BLOCK*.

## 30.39 WMAN$

| | |
|---|---|
| Syntax | WMAN$ |
| Location | TinyToolkit, BTool |

This function returns the version number of the Window Manager. If no Window Manager is present, WMAN$ returns an empty string.

**Example 1**

SCR_SIZE is incompatible with the Window Manager because the channel definition blocks for windows are different from those used when no Window Manager is present, causing SCR_SIZE to return wrong values or produce errors. But calculating the result of SCR_SIZE is so simple that it can be replaced by a BASIC procedure to be used whenever the Window Manager is detected. w_width and w_height define the window size.

```
100 IF LEN(WMAN$) THEN
110   size=8+w_width*w_height/8
120 ELSE size=SCR_SIZE
130 END IF
```

**Example 2**

Non-destructible windows can be simulated by programs if there is no Window Manager present to take over that work.

```
100 OPEN#3,con_200x50a100x50
110 IF WMAN$="" THEN ScrTmp=S_SAVE(#3)
120 BORDER#3,1,4: PAPER#3,3: CLS#3

...... (main program using #3) ...

800 CLOSE#3
810 IF WMAN$="" THEN S_LOAD ScrTmp
820 STOP
```

**CROSS-REFERENCE**

*QRAM$* returns the version number of the Pointer Interface.

## 30.40 WMON

| | |
|---|---|
| Syntax | WMON [mode] or<br>WMON [mode] [, xoff] [, yoff](SMS Only) |
| Location | THOR 8, THOR XVI, Toolkit II |

When the QL is first started up in Monitor mode, the windows #0, #1 and #2 are opened in the following sizes and positions, with the following borders:-

- #0 is con_512x50a0x206 (no border)
- #1 is con_256x202a256x0 (BORDER #1,1,7,0)
- #2 is con_256x202a0x0 (BORDER #2,1,7,0)

As with WTV, this command resets the three default windows to the above sizes, positions and borders. If one parameter is passed, this will alter the screen MODE.

The second variant allows you to move the SuperBASIC windows, by specifying an offset which will be used to calculate the top left hand position of the windows. If only one parameter (other than the MODE) is specified, then this will be taken to be both the x and y offset, otherwise you can specify both. This will only work on higher resolution displays. Also, if the second variant is used, if an outline has previously been defined (for example with OUTLN), then the contents of the three windows will be retained and moved to the new position - this is equivalent to following the WMON command with an OUTLN command with the details of the new position and size.

**Example**

```
WMON 4
```

Will reset standard windows and set MODE 4.

```
WMON , 50
```

Resets the standard windows, in current MODE. The windows are set as follows:

- #0 is con_512x50a50x256(BORDER #0,1,7,0)
- #1 is con_256x202a306x50(BORDER #1,1,7,0)
- #2 is con_256x202a50x50(BORDER #2,1,7,0)

```
WMON 4,50,50
```

Is the same except it forces MODE 4.

**NOTE 1**

WMON does not reset the PAPER and INK colours of the three windows.

**NOTE 2**

On some versions of Minerva (pre v1.78) and Toolkit II, if you do not specify the mode, this command will have no effect.

**NOTE 3**

On versions of the THOR 8 (pre v4.01) #0 appeared one pixel too far up the screen following WMON.

**NOTE 4**

On SMS prior to v2.53 WMON would set an OUTLN if one had not already been set.

**SMS NOTE**

As well as adding the second variant, SMS adds a border to #0 (see example above). v2.67+ has also fixed various problems with this command.

**CROSS-REFERENCE**

Also see *WTV*, *WM*, *WSET*, *WMOV* and *MODE*.

## 30.41 WMOV

| | |
|---|---|
| Syntax | WMOV [#] channel [!] |
| Location | PEX (v20+) |

This command allows you to interactively alter the size and position of the specified Window channel by using the following keys:

- <cursor keys> Move the Origin.
- <SHIFT><cursors> Alter the size of the Window. (See below)
- <ESC> Leave the procedure - do not alter Window size and position.
- <ENTER> Accept the new size and position.

Note that <ALT> plus the <cursor keys> or <SHIFT><cursors> allows you to move more quickly.

You can use this command to re-size a specified BASIC window (use # before channel) or a window used by another Job. If you wish to do the latter, then you will need to omit the # and channel must be the QDOS Channel number (see CHANNELS). PEX22 onwards ensured that when you use this command to alter the size and position of the primary window of a job (set with OUTL), the sizes and relative origins of all secondary windows are preserved. PEX22 onwards also allows you to place an exclamation mark (!) after the channel number, in which case the window sizes cannot be altered - only their position.

**WARNING**

Do not press <CTRL><C> or change Jobs whilst using this command - it can crash the system!!

**CROSS-REFERENCE**

Also see *WTV*, *WMON*, *PICK%*, and *OUTL*.

## 30.42 WPUT

| | |
|---|---|
| Syntax | WPUT [#ch\position,] [item $^{*}$[,item$^{i}$]$^{*}$ ..] or<br>WPUT [#ch,] [item $^{*}$[,item$^{i}$]$^{*}$ ..] |
| Location | SMSQ/E |

This command is very similar to BPUT, except that this sends a word (in the range 0..65535) to the given channel (default #3).

**CROSS-REFERENCE**

See *WGET* and *BPUT*.

## 30.43 WREN

| | |
|---|---|
| Syntax | WREN [#ch,] [wild1] [TO wild2] |
| Location | Toolkit II |

This command allows you to rename several files at the same time. It allows wildcards on both the source and destination parameters. If the source parameter (wild1) does not include a valid device, the default data device will be used. However, the way in which wild2 is calculated, is even more complex than normal:

1. If wild2 is not specified, rename each file using the default destination directory.
2. If wild2 is specified and contains a device, use that device.
3. If wild2 does not include a device, use the same device as for wild1 (ie. the device specified as part of wild1 or DATAD$).

Beyond this, WREN acts in a similar way to WCOPY, listing each file that is being renamed to the specified channel (default #0). However, instead of moving the old file, the header is merely amended to reflect the new name.

**Examples**

```
WREN flp1_QUILL_ TO flp1_
```

could be used to take all of the Quill files out of a sub-directory into the main directory, by deleting the sub-directory prefix.

```
DEST_USE flp1_QUILL_
DATA_USE flp1_
WREN
```

would have the opposite effect.

**NOTE**

Any attempt to rename a file across to a different device will report the error 'Bad Name'.

**CROSS-REFERENCE**

*RENAME* renames one file at a time. *WCOPY* contains details of wildcards.

## 30.44 WSET

| | |
|---|---|
| Syntax | WSET type [,mode] |
| Location | ATARI_REXT |

This command resets the windows #0, #1, and #2 to a pre-defined size and position. There are a set of eight definitions built into the Emulator, which can be chosen by setting type to a value in the range 0…7.

```
WSET -1
```

will reset the three windows to the size and positions specified with the WSET_DEF command. If the optional parameter mode is supplied, this will alter the display mode to that specified, otherwise, the screen mode remains unchanged.

**CROSS-REFERENCE**

*WMON* and *WTV* are similar commands under Toolkit II. Normally, you would use *MODE* to alter the screen mode only. See also *WSET_DEF*.

## 30.45 WSET_DEF

| | |
|---|---|
| Syntax | WSET_DEF x0,y0,a0,b0, x1,y1,a1,b1, x2,y2,a2,b2 |
| Location | ATARI_REXT |

The command WSET_DEF allows you to set up a user-defined size and position for each of the three default windows, #0, #1 and #2. Each set of four parameters is used to specify the size x,y and position (a,b) of each window.

**Example**

```
WSET_DEF 448,40,32,216, 448,200,32,16, 448,200,32,16
WSET -1,8
```

is the same as WTV 8

**CROSS-REFERENCE**

See *WSET*.

## 30.46 WSTAT

| | |
|---|---|
| Syntax | WSTAT [#ch,] [wild] or<br>WSTAT \file [,wild] (Toolkit II only) |
| Location | Toolkit II, THORs |

The command WSTAT works in a very similar way to WDIR except that alongside the filenames, it lists the length of each file and the update time.

**Example**

```
WSTAT QUILL_
```

will produce a list of all of the files on the data device which are in the QUILL sub-directory.

**NOTE**

In current versions of Toolkit II (up to v2.85 at least), WSTAT cannot cope with the DEV device where this is pointing to a sub-directory.

**CROSS-REFERENCE**

*DIR* will produce a list of all of the files on a given medium. *WCOPY* contains details of how wildcards operate.

## 30.47 WTV

| | |
|---|---|
| Syntax | WTV [mode] or<br>WTV [mode] [, xoff] [, yoff](SMS Only) |
| Location | THOR 8 (v4.20+), THOR XVI, Toolkit II |

When the QL is first started up in TV mode, the windows #0, #1 and #2 are opened in the following sizes and positions, without any borders:-

- #0 is 448x40a32x216
- #1 is 448x200a32x16
- #2 is 448x200a32x16

Whilst testing programs, it is all too easy for these three windows to be redefined (especially #1 which is the default window). The command WTV allows you to easily set those three windows to their default size and position as well as taking an additional parameter for setting the mode in the same way as the MODE command (default MODE 4).

Any border attached to each window is switched off, except under SMS (see below). Also, if the second variant is used, if an outline has previously been defined (for example with OUTLN), then the contents of the three windows will be retained and moved to the new position - this is equivalent to following the WTV command with an OUTLN command with the details of the new position and size.

**NOTE 1**

WTV does not reset the PAPER and INK colours of the three windows.

**NOTE 2**

On some versions of Minerva (pre v1.78) and Toolkit II, if you do not specify the mode, this command will have no effect.

**NOTE 3**

On SMS prior to v2.53 WTV would create an OUTLN if one does not exist.

**SMS NOTE**

The SMS version of the command adds a border to #0, #1 and #2 (as with WMON) and also allows you to reposition the main windows (see WMON). v2.67+ also fixed several problems with this command.

**CROSS-REFERENCE**

Also see *WMON*.

## 30.48 W_CRUNCH

| | |
|---|---|
| Syntax | W_CRUNCH ( #channel, colour ) |
| Location | Windows (DIY Toolkit - Vol W) |

This toolkit is designed (like the SuperWindow Toolkit) to provide you with facilities for storing parts of the QL's screen in memory so that you can recall them at a later date, thus providing the QL with non-destructible windows inside programs.

Whilst the Pointer Environment provides programs with non- destructible windows, this only ensures that when a program ends, the area of the screen which was occupied by that program is restored so that it looks the same as when the program started. Also, when you switch to another program, the whole of that program's display area appears on screen, overwriting anything else (see OUTLN) - the display covered by the newly activated program is then stored in memory to be recalled at a later date. However, unless you use specific functions (for example those supplied as part of the Qptr Toolkit, or supplied with this toolkit), if a program OPENs one window over the top of another window owned by that program, when that second window is CLOSEd, the area underneath is not restored (see the example below).

This function allows you to store the area under a specified window channel in memory in a compressed form. Ideally the window should be a number of pixels wide which is divisible by eight and also have its left boundary (after taking any BORDER into account) on a pixel which is divisible by eight (if not then this function will store a slightly larger area of the screen than that covered by the window). This function compresses the screen by reference to the colour parameter - this should either be 4 to store the green pixels or 2 to store the red pixels.

The function is therefore only really of use in MODE 4 since other MODEs may use a lot more colours. Other pixels are ignored and will therefore not be copied back onto the screen with W_SHOW. Since most screens have text in one colour on top of another background, this function is ideal for those circumstances. This function is also very useful for storing Icons and other symbols, since the image, once stored with this function, can be copied back to the screen with W_SHOW again and again. The value returned by W_CRUNCH is the address of the area in memory where the copy of the screen is stored - you will need to keep this address for use by the other functions in the toolkit.

**Example**

Try the short program which follows and note how when you press <ENTER> to close the temporary window, the display does not alter:

```
100 OPEN #2,con_448x200a32x16: PAPER #2,0: CLS #2: INK #2,2
110 FOR i=1 TO 15
120   PRINT #2, 'This is window #2 - Line number '; i
130 END FOR i
140 INK #2,4: PRINT #2,'PRESS A KEY TO OPEN TEMPORARY WINDOW'
150 PAUSE
160 OPEN #3,con_230x40a80x100: PAPER #3,2: CLS #3
170 INK #3,7: PRINT #3, 'This is a temporary window'
180 INPUT #3, 'Press <ENTER> to close this window ';a$
190 CLOSE #3
```

Instead, you can use W_CRUNCH to store #2 and then restore it once #3 has been closed - add the following lines:

```
155 base=W_CRUNCH(#2,2)
200 CLS #2
210 W_SHOW #2,base
```

Note how only the characters which were printed in Red Ink were stored. You could have just stored the area under the temporary window by taking the original example and adding the lines:

```
160 OPEN #3,con_230x40a80x100: PAPER #3,2
165 base = W_CRUNCH(#3,2): CLS #3
185 PAPER #3,0: CLS #3
187 W_SHOW #3,base
```

Note the need to store the contents of the window with W_CRUNCH before it is cleared with CLS !!.

**NOTE 1**

This function will only work on screen resolutions of 512x256 pixels.

**NOTE 2**

The memory used by the function will be reclaimed by CLCHP, or LOAD, LRUN or NEW. You can also use DISCARD address or RECHP address+4 to remove it specifically (although note the different address requirement for RECHP).

**CROSS-REFERENCE**

See *SCR_REFRESH* and *SCR_STORE*. See also *W_STORE*, *W_SHOW*. *W_SWAP*, *SET_RED* and *SET_GREEN* allow you to recolour windows.

## 30.49 W_SHOW

| | |
|---|---|
| Syntax | W_SHOW #channel, address |
| Location | Windows (DIY Toolkit - Vol W) |

This command takes an image stored at the specified address using either the W_CRUNCH or W_STORE functions and then copies it across to the specified window channel.

**NOTE 1**

This command will only work on screen resolutions of 512x256 pixels.

**NOTE 2**

The memory used by W_CRUNCH or W_STORE is not released, so that you can re-display the screen again in the future.

**NOTE 3**

An out of range error will be reported if the stored image will not fit within the specified window.

**CROSS-REFERENCE**

See *SCR_REFRESH* and *SCR_STORE*. See also *W_STORE*, *W_CRUNCH*. *W_SWOP*, *SET_RED* and *SET_GREEN* allow you to recolour windows.

## 30.50 W_STORE

| | |
|---|---|
| Syntax | W_STORE ( #channel ) |
| Location | Windows (DIY Toolkit - Vol W) |

This function is very similar to W_CRUNCH except that it stores the whole of the contents of the specified window (not in compressed form). It also stores all of the colours, not just green or red.

**NOTE**

Refer to the notes for W_CRUNCH.

**CROSS-REFERENCE**

See *W_CRUNCH*!

## 30.51 W_SWAP

| | |
|---|---|
| Syntax | W_SWAP #channel |
| Location | Windows (DIY Toolkit - Vol W) |

This command looks at the specified window channel and swaps over red and green bits on the display, effectively changing the colours on screen.

**NOTE 1**

This command will only work on screen resolutions of 512x256 pixels.

**NOTE 2**

This command should not really be used in MODE 8.

**CROSS-REFERENCE**

*W_SWOP* is exactly the same. *RECOL*, *SET_RED* and *SET_GREEN* also allow you to recolour a window. Refer to the QL display Appendix.

## 30.52 W_SWOP

| | |
|---|---|
| Syntax | W_SWOP #channel |
| Location | Windows (DIY Toolkit - Vol W) |

This command is exactly the same as *W_SWAP*.

CHAPTER

THIRTYONE

# KEYWORDS X

## 31.1 XCHANGE

| | |
|---|---|
| Syntax | XCHANGE (a1 TO a2, c1 TO c2) |
| Location | BTool |

The function XCHANGE replaces all occurrences of byte c1 by c2 from memory locations a1 to a2 inclusive. The function counts the exchanged bytes and returns the sum.

**Example**

Provided that you have enough free memory, this small program replaces line-feed characters CHR$(10) by carriage returns CHR$(13) in ram1_test_txt as quickly as possible:

```
100 ch=FILE_OPEN("ram1_test_txt",0)
110 IF ch<0 THEN REPORT ch: STOP
120 length=FLEN(#3)
130 memory=ALCHP(length)
140 IF NOT memory THEN REPORT -3: STOP
150 x=LOAD_FILE(#ch,memory,length)
160 IF XCHANGE(memory,memory+length-1,13,10) THEN
170   GET#ch\0
180   SAVE_FILE#ch,memory,x
190   TRUNCATE#ch
200 END IF
210 CLOSE#ch: RECHP memory
```

**CROSS-REFERENCE**

*SEARCH*, *COPY_B*, *COPY_W*, *COPY_L*

## 31.2 XDRAW

| | |
|---|---|
| Syntax | XDRAW x1,y1 TO x2,y2 |
| Location | HCO |

The command XDRAW draws lines, just like LDRAW, but it draws the lines in white ink, using the XOR mode; so drawing the same line will remove it again without changing the background.

**Example**

```
100 REPeat scan 110 w% = RND(1 TO 200)
120   FOR x% = 0 TO 511 + w%
130   IF x% < 512 THEN
140     XDRAW x%,0 TO x%,255
150   END IF
160   IF x% > w% - 1 THEN
170     XDRAW x%-w%,0 TO x%-w%,255
180   END IF
190   IF KEYROW(1)&&8 THEN EXIT scan
200   END FOR x%
210 END REPeat scan
```

**CROSS-REFERENCE**

*LDRAW*, *LINE* with *OVER*.

## 31.3 XLIM

| | |
|---|---|
| Syntax | XLIM or<br>XLIM #ch (v2.08+) |
| Location | ATARI_REXT (v1.29+) |

This function returns the horizontal size of the screen in pixels. It can therefore be used to ascertain if the Extended Mode-4 is present, and if so, the size of the screen available to the program (ie. 512x256, 768x280 or larger!). The second variant makes this function the same as SCR_XLIM.

**Example**

A program may wish to use the whole of the screen for its output, adapting itself accordingly:

```
100 MAX_WIDTH=XLIM
110 MAX_HEIGHT=YLIM
120 OPEN #1,'CON_' & MAX_WIDTH & 'x' & MAX_HEIGHT & 'a0x0'
```

**NOTE**

The Pointer Interface must be present in order for XLIM to work.

**CROSS-REFERENCE**

*YLIM* returns the maximum screen height. *QFLIM* and *SCR_XLIM* are very similar. Use *QRAM$* or *WMAN$* to see if the Pointer Environment is available.

## 31.4 XOR

| | |
|---|---|
| Syntax | condition1 XOR condition2 |
| Location | QL ROM |

This combination operator combines two condition tests together and will have the value 0 if both condition1 and condition2 are true or both are false or 1 if either condition1 or condition2 are true (but not both).

Please note the difference between this and the bitwise XOR operator: x^^y, which compares x and y bit by bit.

**Examples**

```
PRINT 1 XOR 0: REMark Returns 1.
PRINT 2 XOR 10 REMark Returns 0.
```

Compare:

```
PRINT 2^^10
```

which returns 8.

```
10 FOR x=1 TO 5
20   FOR y=1 TO 5
30     IF x=3 XOR y >1 AND y<3:PRINT x;'=>';y,
40   END FOR y
50 END FOR x
```

produces the following output:

```
1=>2 2=>2 3=>1 3=>3 3=>4 3=>5 4=>2 5=>2
```

**CROSS-REFERENCE**

*AND*, *OR* and *NOT* are the other combination operators.

## 31.5 X_PTR%

| | |
|---|---|
| Syntax | X_PTR% |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I), Amiga QDOS v3.20+ |

This function will return the x co-ordinate of the pointer which is controlled by the mouse. The value is given in absolute pixel co-ordinates, with the point 0,0 being the top left hand corner of the screen.

**CROSS-REFERENCE**

See *PTR_LIMITS* and *PTR_ON*. *Y_PTR%* reads the y co-ordinate.

CHAPTER

THIRTYTWO

# KEYWORDS Y

## 32.1 YEAR%

| | |
|---|---|
| Syntax | YEAR% [datestamp] |
| Location | SMSQ/E |

This function complements the *DATE* and *DATE$* functions, by returning the year number corresponding to the given datestamp, or current date, if no datestamp was given.

**Examples**

```
PRINT YEAR% (0)
```

will print the year part of the QL's epoch, 1961

```
PRINT YEAR%
```

will print the current year number.

**CROSS-REFERENCE**

See *DATE*, *MONTH%*.

## 32.2 YLIM

| | |
|---|---|
| Syntax | YLIM or<br>YLIM #ch (v2.08+) |
| Location | ATARI_REXT (v1.29+) |

This function returns the vertical size of the screen in pixels. The second variant makes this function the same as SCR_YLIM.

**NOTE**

The Pointer Interface must be present for this function to work.

**CROSS-REFERENCE**

See *XLIM* for details.

## 32.3 Y_PTR%

| | |
|---|---|
| Syntax | Y_PTR% |
| Location | KMOUSE, MOUSE (DIY Toolkit - Vol I), Amiga QDOS v3.20+ |

This function will return the y co-ordinate of the pointer which is controlled by the mouse. The value is given in absolute pixel co-ordinates, with the point 0,0 being the top left hand corner of the screen.

**CROSS-REFERENCE**

See *PTR_POS* and *PTR_MAX*. *X_PTR%* reads the x co-ordinate.

CHAPTER

# THIRTYTHREE

# KEYWORDS Z

## 33.1 ZAP

| | |
|---|---|
| Syntax | ZAP keyword$ |
| Location | TinyToolkit |

This command removes a given keyword from the name table so that SuperBASIC is no longer aware of its existence. The code remains in memory so no memory is freed. A ZAPped keyword cannot be recovered without re-loading the code or resetting the system.

**Example**

You try to run a SuperBASIC program but it stops at the following line:

```
1120 er=2: es=.9: ET=1.4: eu=0
```

with error -17. The author used et as a variable because the ET command was not present when (s)he wrote that program. The more resident keywords that are present, the more it is likely that such collisions may occur. ZAP "ET" and re-loading the program will cure the problem.

**NOTE**

It is okay to ZAP incompatible and bug-ridden keywords, but removing essential keywords like ED, EDIT, AUTO, LIST to stop the user from editing a program will decrease the QL's multitasking abilities. Yes, multitasking depends on this general rule: the more a program influences the whole system and may affect other programs, the less the computer can multitask.

**CROSS-REFERENCE**

Keywords can be renamed with *NEW_NAME*. *KEY_RMV* works in the same way as *ZAP*. See also *TINY_RMV*.

CHAPTER

THIRTYFOUR

# KEYWORDS OTHER

## 34.1 _DEF%

| | |
|---|---|
| Syntax | _DEF% [ (#channel) ] |
| Location | DEFS (DIY Toolkit - Vol A) |

This function creates a table of all of the SuperBASIC PROCedures and FuNctions used within the program currently in memory. The table appears in the specified CONsole channel, if any (default #2). You can use the cursor keys to highlight the required PROCedure or FuNction name and then press <ENTER>, in which case the function will return the line number of the program line which contains the relevant DEFine PROCedure or DEFine FuNction. If you press the <ESC> key an 'Incomplete' error is caused. If no PROCedures or FuNctions are defined, then a 'Not Found' error is reported.

**Example**

```
ED _DEF%
```

**CROSS-REFERENCE**

*_DEF$* and *_NAME$* are similar. *CODEVEC* returns the machine code base address of a Machine Code Procedure or Function.

## 34.2 _DEF$

| | |
|---|---|
| Syntax | _DEF$ [ (#channel) ] |
| Location | DEFS (DIY Toolkit - Vol A) |

This function is similar to _DEF% in that it creates a table of all of the SuperBASIC PROCedures and FuNctions used within the program currently in memory. The table appears in the specified CONsole channel (if any - default #2). You can use the cursor keys to highlight the required PROCedure or FuNction name and then press <ENTER>, in which case the function will return the selected name of the PROCedure or FuNction. If you press the <ESC> key an 'Incomplete' error is caused. If no PROCedures or FuNctions are defined, then a 'Not Found' error is reported.

**CROSS-REFERENCE**

See *_DEF%*

## 34.3 _NAME$

| | |
|---|---|
| Syntax | _NAME$(offset) |
| Location | DEFS (DIY Toolkit - Vol A) |

This function can be used to examine the SuperBASIC name list, which contains the names of all machine code Procedures, Functions, variables, SuperBASIC PROCedures and SuperBASIC FuNctions which are available to SuperBASIC.

**Example**

A program to print out the full name list (this only works on Minerva and SMSQ/E):

```
100 nlist_start=PEEK_L (\\ HEX('20'))
110 nlist_end=PEEK_L (\\ HEX('24'))
120 nlist_len=nlist_end-nlist_start
125 names=0
130 FOR i=0 TO nlist_len
140   x$=_NAME$(i)
150   names=names+1+LEN(x$)
155   IF names>nlist_len: EXIT i
160   PRINT i,x$
165   PAUSE
170 END FOR i
```

**CROSS-REFERENCE**

The name list can be tidied up with *CLEAR*. *EXTRAS* will list all the machine code Procedures and Functions. See also *LOOKUP%*.

CHAPTER

THIRTYFIVE

# APPENDICES INTRODUCTION

This section provides a brief introduction to each Appendix, explaining the background. Each heading is preceded by the number of the Appendix in square brackets.

**[1]** *A1. Minerva*

This is an introduction to the Minerva ROM replacement which not only corrects many bugs contained in the original Sinclair ROMs, but additionally introduces completely new features to SuperBASIC and in general as well as speeding up the operation of the QL.

It is worth noting that the QDOS operating system had relatively few bugs, whereas in the early QL days, hackers competed to find as many bugs in the SuperBASIC interpreter as possible.

**[2]** *A2 SMSQ/E*

This is an introduction to the SMSQ/E replacement operating system which is available for several different hardware platforms, from the standard QL to QPC, a software emulator for PCs.

This is a massive extension to the original QDOS operating system with several new keywords and fixes of existing bugs.

**[3]** *A3 Emulators*

This gives general information about the range of emulators which are available to allow QL software to run on other computers.

**[4]** *A4 Thor Computers*

This provides some information on using SuperBASIC on the THOR range of computers.

**[5]** *A5 Expansion Boards*

This section gives a general introduction to the main expansion boards currently available for the Sinclair QL. Various older add-on boards are available second-hand which really only added Toolkit II, extra memory and possibly disk access to the QL. The boards listed here actually replace parts of the original QL, increasing speed and adding a host of other features.

**[6]** *A6 Compatibility*

This section provides basic and some expert background information on software and hardware compatibility across the range of different QL machines available.

**[7]** *A7 Multiple Basics*

This section provides details of the various QL implementations which allow you to run several BASIC programs at the same time, as well as detailing how multiple BASICs should be used.

**[8]** *A8 Error Messages*

This section lists the QDOS error messages on the different ROMs, or at least those where we could get hold of the text. The main section will usually refer to the error code, so this appendix allows you to find out about how this is represented on your ROM as well as giving some general advice concerning what may have caused the problem.

**[9]** *A9 Character Set, Keyboard*

This lists the character set and informs you about the different keys needed to achieve the same character code on the different QDOS computers. Again, this may be incomplete, but it helps the programmer recognise foreign keyboard layouts.

**[10]** *A10 Designing New Character Sets (Fonts)*

This section explains how you can create and link new fonts into the QL.

**[11]** *A11 Mathematics*

This section gives some necessary background to the QL mathematics package to make certain keywords easier to use. Do not worry, it is understandable (we hope!).

**[12]** *A12 Device Drivers*

This forms an introduction to device drivers in general and also provides numerous detail about the different drivers and hardware on different machines.

**[13]** *A13 Extended Pointer Environment*

This section provides a short introduction to the Pointer Environment and explains what it has to offer.

**[14]** *A14 Coercion*

This provides some brief details as to how the QL converts strings to numbers if necessary.

**[15]** *A15 Mouse Drivers*

This section provides details of the various ways of linking a Mouse to the various QL implementations.

**[16]** *A16 The QL Display*

This section provides various details concerning how the QL's display is arranged in memory and how to ensure that programs are able to work on all the various display resolutions available to the QL.

**[17]** *A17 Networks*

This section details the various options which are available to allow the QL to connect to other computers over Networks as well as setting out various details about how networks work and how to ensure greater reliability over data sent.

**[18]** *18 Configuring Programs*

This section covers the use of the CONFIG utility to configure a program with suitable default settings.

CHAPTER

THIRTYSIX

# A1. MINERVA

## 36.1 A1.1 INTRODUCTION

Minerva is a brand new operating system which has been designed for the QL by QView, parts of which are already incorporated in QDOS emulators.

Based upon the existing QL Roms, it has now developed into a system which does not rely upon any of the existing code and thereby side-steps problems with copyright. However, due to the new coding, there are bound to be one or two slightly grey areas where compatibility must meet with compromise. From the SuperBASIC point of view, there are no real problems with using programs written for earlier ROMs (possibly excluding FB Roms which insisted on using AT y,x instead of AT x,y), and Minerva merely enhances this superb programming language to provide the SuperBASIC programmer with one of the most flexible and quickest implementations of BASIC ever devised.

However, there are one or two enhancements made by Minerva which are unavailable on earlier ROMs, which of course will limit the portability of programs which use these advanced features (unless the SuperBASIC program is compiled with QLiberator or Turbo).

The idea of this section is therefore just to point out some of the possible pitfalls with which programmers will be faced when writing SuperBASIC which is specifically designed to run on all versions of Minerva.

**WARNING:**

v1.98 of Minerva caused more problems than it solved and you should therefore obtain v1.97 if you have this version of the operating system.

## 36.2 A1.2 Windows and Closing Windows

A problem exists with some versions of Minerva and Lightning SE. The problem appears to be that if Lightning is switched on, and then windows are opened on screen, unless they are closed in the reverse order to which they were opened, they will remain visible on screen despite NEW or MODE commands until another window with the same channel number is opened.

Example:

```
100 OPEN#3,scr_448x200a32x16: PAPER#3,2: CLS#3
110 OPEN#4,scr_448x200a32x56: PAPER#4,4: CLS#4
120 CLOSE#3: CLOSE#4
```

This program will cause the problem and you will be unable to remove either #3 or #4 from the screen until they are re-opened!

To avoid the problem, alter line 120 to read:

```
120 CLOSE#4: CLOSE#3
```

QJump's Pointer Environment prevents this problem occurring and this can be regarded as a standard system addition.

## 36.3 A1.3 Dual Screen Mode

Perhaps the greatest problem with programs written on earlier ROMs is that they (or machine code extensions used by them) tend to assume that the screen display will always begin at 131072 ($20000) and that the system variables will begin at 163840 ($28000). When Minerva is started-up, the user can opt to run in Dual Screen Mode, which makes two screens available to the programmer, the first of which is located at the standard display address (131072 - $20000 in hexadecimal) and the second of which is located on top of the old system variables address (163840 - $28000 in hexadecimal) - the system variables are moved out of the way. Programs can run on either of the screens and should be written so that they make no assumptions about the start address of the screen.

Minerva allows the user to alter which screen a program will run on before the program is actually loaded, by using the command MODE 96,-1, and therefore the programmer has no control in general over which screen will be used.

This problem can also exist with the THOR XVI which allows several screens to be in operation at any one time.

Various functions exist to find the start address of the current screen (eg. SCREEN) and also to find the start address of the system variables, (eg. SYS_BASE). These functions should be used whenever possible.

The other main problem is that in current versions of Minerva, when a window is opened on screen, Minerva attempts to open it in the MODE of the currently displayed screen, rather than the mode of the screen on which it will be open. This can lead to various problems, such as illegible writing and flashing on screen. The answer to this is to ensure that the current screen when the program is started is also the displayed screen - see MODE.

Other problems exist when using the dual screen mode with both Lightning and the Pointer Environment, which can make it extremely difficult to change MODEs on only one screen in the dual screen mode. At the moment there is no answer to this, other than to pester Digital Precision and Qjump to produce compatible products. There may also be similar problems with Speedscreen, but we have been unable to test this.

The other remaining problem is that current versions of Turbo and Supercharge do not support Minerva's dual screen mode, and programs compiled with either of these packages will not run in the dual screen mode.

A problem which users with printers may or may not have noticed, is that Minerva tends to lose characters on their way to the printer when in dual screen mode.

## 36.4 A1.4 Border

On some early Minerva ROMs (and on one or two of the THOR ROMs), the BORDER command will not work correctly unless a parameter is supplied.

The original ROM accepted BORDER on its own to switch off the border on #1 (although this was not pointed out by the QL Manual). It is therefore essential that to retain compatibility BORDER 0 is used.

## 36.5 A1.5 Empty Brackets

An empty bracket is always regarded as a syntax error by original QL ROMs whereas Minerva tolerates them, for example to indicate that a function is called with a parameter, eg. DATE(). If a program should be portable than you have to avoid this style because all ROMs other than Minerva and SMS mark such lines with a MISTake:

```
100 MISTake PRINT DATE()
```

Since DATE() and VER$() are effectively the same as DATE and VER$, it is recommended to use the latter syntax.

### A1.6 INTEGER TOKENISATION

From v1.79 onwards, Minerva has tidied up the storage of numbers in SuperBASIC programs, in that if integer tokenisation is enabled, any numbers contained in a program are stored in as few bytes as possible. Although programs which are LOADed are unaffected by this since the interpreter converts the ASCII characters into the internal format as the program is loaded, certain utilities are unable to cope with integer tokenisation. In particular, the Supercharge compiler and early versions of the Turbo (pre v4.3) and QLiberator (pre v3.32) compilers are unable to compile such programs.

If you use the QLOAD utility from Liberation Software, this stores a program in its internal format so that it can be loaded very quickly. Unfortunately, this does mean that programs which are QSAVEd on Minerva whilst integer tokenisation means that if when the program is QLOADed with integer QSAVEd, you can end up with a mixture of integer tokens and floating point tokens within the same program.

To convert a program under Minerva which has been written with Integer Tokenisation enabled so that it may be compiled with Supercharge or early versions of Turbo, or even QLOADed on a different ROM version, you will need to use something along the lines of:-

```
SAVE ram1_convert_bas
POKE \\212,128
LOAD ram1_convert_bas
```

If you try to QLOAD a program which has been QSAVEd with Integer Tokenisation enabled, on a different ROM, you will notice that various numbers have disappeared from the listing.

## 36.6 A1.7 MultiBASICs

Minerva allows you to have several BASIC interpreters which can all multitask. Essentially, you retain the original SuperBASIC interpreter together with several copies of that interpreter, each of which is known as a MultiBASIC.

A MultiBASIC will in fact operate in the same way as SuperBASIC, and you can link different toolkits with each copy of the interpreter, knowing that they will not be available to the other existing interpreters.

The only problem is that many toolkits have been written with only the original SuperBASIC interpreter in mind, and some commands are therefore unable to access the MultiBASIC's variables. Fortunately, the majority of commands and functions do in fact still work with MultiBASICs, unless used from within a compiled program.

## 36.7 A1.8 Strings

Minerva has altered the way in which strings are handled, in much the same way as SMSQ/E - please refer to DIM for an explanation of the way in which dimensioned and undimensioned strings are dealt with on the various QL implementations.

There was however a problem prior to v1.98 of Minerva in that if you concatenated two strings together to make a string longer than 32764 characters, this could corrupt BASIC. An example of concatenation is:

```
a$ = 'Hello' & b$
```

CHAPTER

THIRTYSEVEN

# A2 SMSQ/E

(Versions 2.78 - 2.95 tested.)

## 37.1 A2.1 Introduction

SMSQ/E is a brand new operating system written by Tony Tebby which has been designed for the QL and emulators. It is currently available for standard QLs, AURORA, QPC (a PC emulator), ST/QL emulators, QXL and QXL2, the Q40 qnd Q60 computers and also the ATARI ST, STE and TT range of computers.

You must take care not to confuse this operating system with some of the earlier forms which have been released, namely SMS2 (a plug in add-on for Ataris) and SMSQ (supplied with the QXL and QXL2 boards) - see the Emulators Appendix for further details.

SMSQ/E is a complete re-write of the original operating system avoiding any copyright problems. However, to run on the original QL, SMSQ/E still requires that a QL operating system (eg. JM or JS) is installed on the chips inside the QL plus a disk drive and a Gold Card (at the minimum). This is because SMSQ/E is disk based, rather than being supplied on ROM chips - you therefore need a means of starting up the QL and loading SMSQ/E from disk. This does not apply to the QXL, QPC and Atari versions which have to load their operating systems from disk anyway.

If you are using SMSQ/E on the Gold Card or Super Gold Card, you should normally use the LRESPR command to start up the copy of SMSQ/E - if this is used in a boot program, if SMSQ/E is already loaded when the LRESPR command to load SMSQ/E is read, then it is ignored. If you use RESPR and LBYTES, then the boot would get into a loop - continually loading the operating system (unless you check VER$='HBA', although this will not work on QXL as SMSQ also reports VER$='HBA').

SMSQ/E also incorporates all of the features of the following (which are available as add on packages to the original QL operating system):

- Toolkit 2
- Pointer Interface
- Qjump's Window Manager
- Hotkey System II
- DEV device driver
- QLOAD / QSAVE from Liberation Software
- OUTLN command.

As an added bonus, Gold Card and Super Gold Card users (including AURORA users) also get the Serial Mouse Driver from Albin Hessler Software which can be loaded into SMSQ/E and allows you to use a mouse plugged into the serial port.

QXL, QPC, Q40/Q60 and Atari users are able to use the mice supplied for their computers (provided that you load in a driver for that computer, eg. the DOS Mouse Driver for IBM Compatible PCs).

SMSQ/E also offers other in-built features:

- Named pipes and a history device
- Improved serial and parallel device drivers
- Level-3 device drivers which allow you to access disks from IBM compatible PCs and Ataris.
- Hi-resolution displays and enhanced colours - refer to Appendix 16

SMSQ/E also includes the SBAS/QD F10 Thing, which allows you to automatically run SuperBASIC programs which are being edited under the QD Editor by pressing F10 (this initiates a Multiple SBASIC which then LRUN's the program). The FileInfo II Thing is also included which allows SuperBASIC programs with filenames ending in _BAS or _SAV to be Executed directly from the Qpac II Files Menu.

Unfortunately, due to the new coding, there are bound to be one or two slightly grey areas where compatibility must meet with compromise.

From the SuperBASIC point of view, there are no real problems with using programs written for earlier operating systems (possibly excluding FB Roms which insisted on using AT y,x instead of AT x,y), and SMSQ/E merely enhances this superb programming language to provide the SuperBASIC programmer with the most flexible and quickest implementations of BASIC ever devised (it is nearly as quick as SuperBASIC compiled with Qliberator).

Some of the enhancements made by SMSQ/E are not available on other operating systems and it is likely that in the future SMSQ/E is the only operating system which will continue to be upgraded. With free upgrades at present there is no reason why you should be using anything but the latest version!

The idea of this section is therefore just to point out some of the possible pitfalls with which programmers will be faced when writing SuperBASIC which is specifically designed to run on SMSQ/E and possibly also on earlier operating systems.

**WARNING:**

Some versions of SMSQ/E appear to have been released in a hurry and are unreliable - most notably, v2.93 and v2.95 which should be avoided.

## 37.2 A2.2 The EOF Function

On early versions of SMSQ/E (pre v2.55) EOF was implemented differently to the original SuperBASIC version, in that it would wait until either data or an end of file code was received from the associated channel. The original would return 1 if no data was waiting in the queue. This version of EOF has now been renamed EOFW.

## 37.3 A2.3 Empty Brackets

An empty bracket is always regarded as a syntax error by original QL ROMs whereas SMSQ/E (in common with Minerva) tolerates them, for example to indicate that a function is called with a parameter, eg. DATE(). If a program should be portable than you have to avoid this style because all other implementations (other than Minerva) mark such lines with a MISTake:

```
100 MISTake PRINT DATE()
```

Since DATE() and VER$() are effectively the same as DATE and VER$, it is recommended to use the latter syntax.

## 37.4 A2.4 Multiple Sbasics

SMSQ/E (in common with Minerva) allows you to have several BASIC interpreters which can all multitask. Essentially, you retain the original SuperBASIC interpreter together with several copies of that interpreter, each of which is known as a SBASIC.

A SBASIC will in fact operate in the same way as SuperBASIC, and you can link different toolkits with each copy of the interpreter, knowing that they will not be available to the other existing interpreters.

The only problem is that many toolkits have been written with only the original SuperBASIC interpreter in mind, and some commands are therefore unable to access the SBASIC's variables. Fortunately, the majority of commands and functions do in fact still work with SBASICs, unless used from within a compiled program.

See Appendix 1 for more details of Multiple BASICs.

## 37.5 A2.5 Improved Interpreter

Unlike the original SuperBASIC, SMSQ/E will look at a program to ensure that all of the structures are validly constructed before it allows you to SAVE or RUN the program. Although this can pick up common programming mistakes (such as forgetting to add END FOR or END DEFine), this can mean that some SuperBASIC programs will now refuse to RUN, reporting an Incomplete Definition. This is especially problematic as SMSQ/E (prior to v2.89) will not allow a single-line PROCedure or FuNction to exist without END DEFine appearing on the line.

Further checks are carried out before a program is RUN or SAVEd and a wide range of error messages have been added to the Interpreter, which may be reported. These error messages are listed in the Errors Appendix.

## 37.6 A2.6 Numbers in Programs

SMSQ/E has extended SuperBASIC by allowing programs to contain both hexadecimal and binary numbers explicitly in the code, such as a=%10 (or a=$02) is the same as a=2. These constructs will result in a MIStake being shown on other implementations (refer to the Mathematics Appendix).

Programs which use this feature can only be compiled using Turbo v4.3 or later.

## 37.7 A2.7 Inbuilt Pointer Environment

SMSQ/E incorporates the Pointer Environment (the files ptr_gen, wman and Hot_rext form part of the operating system and therefore cannot be loaded separately). This means that some programs which could not be run under the Pointer Environment previously, cannot be run under SMSQ/E. As far as we are aware, there are very few programs which cause a problem.

## 37.8 A2.8 Undefined Variables

SMSQ/E differs from all other implementations of SuperBASIC in that it gives values to variables which have not yet been defined. Whereas on other implementations, if you do not have a program in memory (eg. after NEW), and enter the following line:

PRINT a,a$ : PRINT a/10 you would see the display:

* *

on screen, and then the error 'At 0,2 : Error in Expression' would be shown.

Under SMSQ/E, you would see the display:

0

0

as an undefined numeric variable is given the value 0 and an undefined string is given the value of an empty string (the second 0 is the result of a/10 where a=0). No error would be reported.

## 37.9 A2.9 Extended Display

There are very few programs which will not work under SMSQ/E, these are mainly programs (or toolkits) which make assumptions about:

### 37.9.1 A2.9.1 Extra Colours

SMSQ/E v2.98 implements a different display driver for many systems, including 65536 colours on the QXL, QPC and Q40 - see Appendix 16. Because the format of the display is different to the original QL, this may cause incompatibility problems with many of the drawing commands covered in this book which assume the original QL screen format. This can however be overcome by only using these commands or programs which cause problems in the original display configuration of 512x256 pixels, with MODE 4 or MODE 8 colours. Use RMODE to check which colour system is currently in use.

People have also noticed that various programs appear in various multiple colours (not intended by the original author). This tends to be due to the fact that the authors have assumed that non-standard colours will be converted by MODE 4 (for example INK 3 under MODE 4 produces red) or into stipples.

### 37.9.2 A2.9.2 Hi-resolution Displays

Unless you are using an original QL motherboard, SMSQ/E supports higher resolution displays, up to 1600x1200 pixels. This may cause further problems for software and commands which assume the original resolution of 512x256 pixels.

## 37.10 A2.10 Problems

There are very few programs which will not work under SMSQ/E, these are mainly programs (or toolkits) which make assumptions about:

- The location of the screen or system variables
- The size of the QL's screen
- The fact that you cannot overwrite the QL's ROM (and therefore there is no need to ensure that POKE commands are not trying to overwrite part of the ROM
- The location of various parts of the operating system (including machine code routines)

Most of these programs will also have difficulty running on anything other than a standard QL (even Gold Cards mean that programs will face problems if they try to overwrite the QL's ROM).

### 37.10.1 A2.10.1 Lightning/Speedscreen

These two programs cannot be used with SMSQ/E. However, SMSQ/E's screen driver is just as quick.

### 37.10.2 A2.10.2 Toolkit III and System Toolkit

Neither of these toolkits will work with SMSQ/E, but then they do not really add very much to the system!

### 37.10.3 A2.10.3 Serial to Parallel Converters

SMSQ/E has speeded up the serial ports somewhat (making them meet the design specifications) and unfortunately this means that some serial to parallel converters work too slowly and some characters are lost. Try SER_PAUSE or a newer converter (or if your computer has a built-in parallel interface, use that!).

### 37.10.4 A2.10.4 Aurora

Aurora users will really need to use at least v2a.85 of SMSQ/E.

### 37.10.5 A2.10.5 Disk Access

There appears to be problems accessing DD disks (double density) under SMSQ/E after v2.91, in that later versions often report file errors or fail to format these disks. This is a major problem which will hopefully be resolved in later versions.

See also FLP_DENSITY.

CHAPTER

THIRTYEIGHT

# A3 EMULATORS

## 38.1 A3.1 Introduction

Not only are there replacement operating systems for the QL (namely QDOS and Minerva), as well as replacement computers (AURORA, Q40 and THOR {no longer available}), but there have been several emulators produced which allow programs written for the QL to run on various other computers.

When Emulators access the hard disk on the host computer, you do not have to worry about the fact that the hard disk is not in QDOS format - the Emulators cope with this in one of two ways:

- The hard disk has to be partitioned, and one (or more) partitions are set aside for the QL files (FORMAT will only affect the specified partitions), or
- The Emulator creates a large single file on the hard- disk (for example called QXL.WIN) which is equal in size to the size specified with the FORMAT command and then QL files are stored within this huge file. The host computer will only see the one QXL.WIN file. This method is used by QXL and QPC.

It doesn't really matter which of these methods is used, as both protect the PC files from being over-written by QL files.

Currently, there are emulators available for the following computers (see the relevant section of this Appendix):

- Apple Macintosh (Power PCs and 68000 Macs)
- PCs(any with a spare ISA slot, otherwise 486 and Pentiums only)
- ATARI(All models except the Falcon)
- Any computer with an UNIX operating system

The main problem with using emulators is that some emulators (QLAY and Q-Emulator) require a copy of the QL operating system. You can use a copy of Minerva with these emulators (obtainable from TF Services - specify that you need it on disk for use with an emulator) or a copy of the original QDOS ROM. Apart from North America, the copyright on the original QDOS ROM is vested in Amstrad plc. who have stated that it can be supplied with emulators so long as their copyright notice appears and also an acknowledgement is included in the manual. In North America, the copyright is not owned by Amstrad plc. QLAY (at least) includes a copy of the JS QDOS ROM, otherwise, can make your own copy of the QL's operating system (from a standard QL) by using the command:

```
SBYTES flp1_OSROM,0,49152
```

Note that you cannot do this with a Gold Card or Super Gold Card plugged in as these alter the operating system.

## 38.2 A3.2 Apple Macintosh

### 38.2.1 A3.2.1 Q-Emulator

A commercial software emulator available from Daniele Terdina which comes in two versions (v3.0 - for 68000 Macs and v2.1 - for Power PCs). A version which runs on IBM compatible PC's is also available.

Minimum requirements are MacOS v7.0, 4Mb of memory, a colour monitor, a 1.44Mb floppy drive and a copy of the QL's operating system.

The speed of the Emulator is really dependent upon the machine on which it is used - the Power PC version is said to nearly equal the speed of a QXL when used on a MacIntosh with a 100 Mhz RISC chip.

Unfortunately, SMSQ/E will not currently work with the Emulator.

You may also want to obtain a copy of Toolkit II on disk to use on the Emulator.

This emulator provides you with a QL with up to 4Mb of memory, which can multitask alongside MAC programs. However there is currently no support for Level-2 Device Drivers, Network ports or the FLASH / TRA commands. You can read and write to QL floppy disks (DD and HD) and also use the Mac's own hard-disk as a QL hard-disk. Minerva's dual screen mode is also supported, but at present only the standard 512x256 pixel resolution display can be used.

The Emulator has an in-built static and dynamic RAM disk and allows you to use the MAC's serial ports (all standard QL BAUD rates are supported, although it is recommended that you use hardware handshaking).

Some difficulties exist due to the different MAC keyboard - for example an OPTION button is used instead of <ALT> and most MAC's will not recognise more than two keys pressed down at a time.

## 38.3 A3.3 IBM Compatible PCs

### 38.3.1 A3.3.1 QPC and QPC2

Commercial software emulators available from Q Branch and Jochen Merz Software.

QPC and QPC2 both need a 486 processor or better (a 486 SX- 25 minimum is recommended, although a Pentium is better!), 4MB RAM, EGA graphics and DOS 6.xx. Although both programs will work with Windows95, only QPC2 will work in a window under Windows95 (or Windows98 / NT). However, the PC needs to be configured not to use any extended memory handling devices and therefore the user will need to amend the PC's AUTOEXEC.BAT and CONFIG.SYS files.

It allows between 1Mb and 16Mb of memory to be used by the QL operating system and supports resolutions of at least the same standard as QXL (v1.40 allows up to 1600x1200 and MODE 8). QPC2 will even stretch the QL screen resolution to fit the PC screen, which can make the QL characters appear much larger than usual.

These emulators are quick (current versions are as fast as the QXL) and have faster disk, serial and parallel port access than the QXL. They do however lack some of the QXL's extra hardware facilities, namely QL compatible Network ports (although users can use SERNET to connect to the QL).

One of the main problems is that the more memory allocated to the Emulator, the slower its disk access - this is because of the slave blocks used to store the contents of files so that if you load a file a second time, it loads much more quickly (see DEL_DEFB). As the QL uses all unused memory as slave blocks, this can slow down initial disk access (where the file has not been read before).

Compared with QXL, QPC and QPC2 do have their advantages also, including the ability to access the PC's colour palette (and thereby dictate which colours may be used in the QL modes), although this will

hopefully not come into the equation when the new colour drivers are released for SMSQ/E. Also, both versions include commands to access the PC's CD ROM drive, although at present only audio CDs are supported (see CD_PLAY and related commands).

QPC and QPC2 also allow you to have more than one QL 'hard disk' on the same PC hard disk, by using a different filename for the QL hard disk (other than QXL.WIN as mentioned above).

Both programs also come complete with the SERNET driver to allow you to use the PC's serial (COM) ports as Network devices - this is however somewhat limited as most PC's only have two such ports (and one is used for the Mouse)! If you need more serial ports for a PC, please contact us, as we have boards which can link up to 198 serial ports to a PC!!

The main problem with earlier versions of the emulator is with loading screen images direct - see LBYTES.

**WARNING**

Do not allocate the whole of the PC's memory to QPC as this can cause a disaster!!

### 38.3.2 A3.3.2 QXL II

This is a plug-in emulator (hardware) now available from Q Branch. QXL II is based on a cut-down version of the 68040 chip which is extremely quick and because it only uses the PC for access to display, keyboard and disk drives, it can co- exist with other PC programs and its speed is not dependent on the main processor speed of the PC.

There was an earlier, slower version of QXL sold by Miracle Systems Ltd. which can have between 1M and 8M of memory.

The QXL boards simply plug into a standard 8 or 16 bit ISA slot on the PC and are one of the fastest versions of the QL currently available (including the original!!). They have 8M RAM in-built and run completely independently from the PC, just using the PC's keyboard, display and disk facilities. QXL even has QL compatible Network ports.

Unfortunately, there are few portable IBM compatible computers with ISA slots and therefore if you wish to use an Emulator on a portable, you will probably need to use one of the two software Emulators.

The main problem with the QXL is that it is fairly slow when accessing the PC's floppy disk drives and serial / parallel ports. Also, users have reported that the mouse response and screen re-draw are fairly sluggish if you run the QXL in a DOS Window under Windows95. It is therefore recommended that you only use QXL under a standard DOS window.

Although the QXL has its own QL compatible Network ports (SMSQ/E users can also use SERNET), some QXL II boards display a few problems and you may need to configure the QXL operating system to change the speed of the network (some machines need it turned down to 24Mhz, others need it turned up to 26Mhz). The Network unfortunately did not work on v2.25 of the QXL software!!

When QXL was first released the software was still undergoing development, and only supported a limited range of commands, lacking a full implementation of SuperBASIC and programs compiled with either Turbo or Supercharge would not run. The majority of programs compiled with Qliberator also had problems. If you have one of these very early versions, you should upgrade - the full version of SMSQ was released for QXLs in March 1995.

QXL comes with its own operating system (SMSQ), but the much improved operating system (SMSQ/E) is now also available for QXLs. SMSQ/E will be needed if you wish to use more than the standard QL's 8 colours on the QXL.

SMSQ as supplied with QXLs comes complete with a copy of Toolkit II (you still need to use TK2_EXT to install the toolkit) and Level-2 Device Drivers. SMSQ can handle three different display resolutions in addition to the standard QL 512x256 screen, if your PC has EGA or VGA graphics. These are 630x350 in EGA mode, 640x480 in VGA mode and 800x600 on most SVGA monitors. These display modes must be configured before the emulator is used - compare SMSQ/E which allows you to change the display at any time using the DISP_SIZE command.

SMSQ adopted a different approach to SMSQ/E in that its main aim was to be as compatible as possible with the original QL, whilst at the same time being quicker than QDOS and incorporating an improved SuperBASIC interpreter (it is very similar to SMSQ/E so far as the interpreter goes). In fact, in the main keywords section of this book, we have referred to SMS meaning both SMSQ and SMSQ/E.

For compatibility reasons, it is not possible for SMSQ to adopt the more advanced drivers or an integrated Pointer Environment such as appear in SMSQ/E. It can however work with the standard PTR_GEN, WMAN and HOT_REXT files which are supplied with most Pointer Environment software and therefore can use the Pointer Environment. In order to have SBASIC set up as an Executable Thing, you will need to enter the command SB_THING on SMSQ after the HOT_REXT file has been loaded.

SMSQ also includes facilities to access IBM compatible disks and the hard-disk on a PC. There were however problems with earlier versions which could not create more than one QL partition on each PC hard-disk and limited each partition to 63 Megabytes (see FORMAT). Even in current versions, if your PC does not support partitioning of hard-disks, you can only have one QL 'hard-disk' on each DOS device - normally C:.

You can overcome this limitation by simply using DOS to rename the QL 'hard-disk' file (QXL.WIN) to something else and then create another QXL.WIN file if you wish to have access to several QL 'hard-disks'. If you do this however, you will need to use DEL_DEFB from the QXL to ensure that it recognises that a new QXL.WIN file is being used.

There were also problems on early versions of SMSQ in recognising when a PC format disk had been swapped for another one and you may get the same DIR listing for both disks. This was however fixed by using either DEL_DEFB or reading the directory of a QL format disk before inserting the second PC format disk.

Lightning and Speedscreen must not be used with QXL, but the screen driver supplied with SMSQ and SMSQ/E is nearly the same speed anyway.

QXL's incorporate an easy means of switching between the QL and the PC - simply press <CTRL><SCROLL-LOCK> to switch out of QL mode and into DOS. This is somewhat limited however, as the PC's display sometimes gets distorted.

One of the problems which remains with QXL is that some users have reported difficulties in FORMATting and writing to QL format HD disks - the problems seem to vary from user to user, and it seems that this may in fact be a problem related to the PC's own hardware.

### 38.3.3 A3.3.3 QLAY

This is a freeware software emulator in its very early stages of development which works on most PCs and will run under either DOS or Linux or even Windows95 (v0.84+). It needs a minimum of a 486 processor running at 66Mhz with 8 Mb of memory in order to work. This is not really a competitor to the two products listed above and may be difficult to use if you've never seen a QL - it is however free and available from the Web on:

http://www.inter.nl.net/hcc/A.Jaw.Venema (This link no lonegr works. NDunbar)

A copy of the JS QDOS ROM is supplied as part of this emulator.

From v0.84+, this emulator will actually allow you to use the QL inside a window under Windows 95 (although this version will not support QL ALTkeys) - all other PC emulators, except for Q-Emulator and QPC2, currently insist on you using a DOS window.

At present it has a few problems in that it has poor error detection and reporting. QLAY cannot currently work with a Mouse and early versions only allowed the standard QL resolution display. From v0.85b (the Windows version), various resolutions up to 1024x768 are supported, with the window being scaled accordingly to fill the PC's screen. Early versions (at least v0.7) did not support QL floppy disks, the PC's serial ports or Networks - it is unknown whether these have yet been added.

QLAY does however allow you to use Microdrives - what it actually does is use a file on either a PC format disk or the PC's hard disk which is identified as a QL microdrive by the extension .MDV - you create a new 'Microdrive' by copying from DOS the file EMPTYDSK.MDV onto the required medium and give it a new name, such as QUILL.MDV. When you start up QLAY (from DOS), you can pass it the names of the two microdrive files it is to use as MDV1_ and MDV2_ and then any files which are SAVEd to MDV1_ (or MDV2_) will be stored as part of the DOS file. For example:

```
QLAY -1QUILL.MDV -2DATA.MDV
```

will allow you to enter the command (inside QLAY) SAVE MDV2_TEST_bas which will then create the QDOS file test_bas inside the DOS file DATA.MDV.

You can also specify whether QLAY is to use up to 8Mb of memory for the QL (although you will need to use Minerva to cope with more than 768K) and even whether microdrives are to be write-protected.

Unfortunately, early versions of QLAY provided no means of getting QL files across to the PC to store in these microdrive files. There is now a separate program (QLAYT) supplied to allow you to do this. A ramdisk is also supplied.

### 38.3.4 A3.3.4 Q-Emulator for Windows95

This is intended to be a shareware Emulator, which again, is in its early stages of development. It is based upon the Emulator of the same name for the Apple MacIntosh and works only under Windows95. It requires a 486 computer at least and supports both QDOS and Minerva (although as with QLAY, you need to obtain a copy of the QL ROM). The Emulator provides the user with up to 4M of memory and the current Alpha version supports the PAR device, QDOS disks and host files.

This Emulator is currently limited to supporting the standard QL display (512x256 pixels); and supports the PC's mouse, and QL BEEP commands (provided that you have PC DirectX drivers). It can use any PC BAUD rate up to 256,000 as well as those supported by the QL.

The TRA command is not supported.

A copy of this Emulator and further details can be obtained from:

http://www.geocities.com/SiliconValley/Heights/1296/winql.html (This link no lonegr works. NDunbar)

## 38.4 A3.4 Atari Computers

There are several hardware based Emulators which are referred to in this book collectively as 'ST/QL Emulator' (excluding SMS2). There is also one software Emulator (SMSQ/E).

## 38.5 A3.4.1 The ST/QL Emulator

This in fact relates to three different QL Emulators which can be fitted to the Atari range of computers. The type of Emulator needed depends upon the Atari computer being used and also when the Emulator was purchased.

All later versions of the Emulators come complete with Atari_rext and AtariDOS toolkits.

### 38.5.1 (a) Atari-QL Emulator

A commercial hardware emulator made by Futura Datasenter in Norway for MEGA ST and 520/1040 machines - this has not been available for some time. Some versions of the emulator supported MODE 8, some did not - it is impossible to check if it does. This only supported the original QL screen resolutions.

### 38.5.2 (b) Extended4-Emulator

A commercial hardware emulator for all ST machines (268,520,1040 but not STE), including STF, STFM and MEGA ST models. It will however not work on the Falcon 030. This may still be available from Jochen Merz Software.

Although this has its own operating system built in, you can upgrade it to SMSQ/E if you wish.

### 38.5.3 (c) QVME

A plug in commercial hardware emulator for Mega STEs and TTs that plugs into the VME slot. It unfortunately will not work with the Falcon 030. This is available from Jochen Merz Software and current versions come complete with SMSQ/E.

This supports a wide range of screen resolutions up to 1024x780 pixels (or theoretically, if you can obtain a monitor, 1024x1024 pixels) are supported. You are also able to choose at runtime (unlike the QL-emulator Extended4) the resolution in which you wish to work, using the DISP_SIZE command - this is only limited by the capabilities of your monitor.

### 38.5.4 In General

Both of the first two hardware emulators must be fitted inside an Atari ST computer and needed a bit of careful soldering to make them work. The QVME simply plugs into the Atari ST.

Once fitted, all of these hardware emulators are based on a JS ROM (unless you have installed SMSQ/E on the QVME emulator); indeed a slightly patched copy of a JS ROM is loaded as the basis for the emulator's operating system; these patches are not documented. The operating system may be loaded from either disk, harddisk or EPROM.

Once loaded, you are presented with the normal QL start-up screen, although later versions of the emulator allow you to start-up by pressing the following:

- F1… MODE 4 + Monitor
- F2… MODE 4 + TV
- F3… Extended MODE 4 + Monitor
- F4… Extended MODE 4 + TV

(On the QVME, only the first two options are displayed).

Together with the image of a JS ROM, the emulator loads in its own set of drivers - please see the section on Drivers. In the latest versions of the emulator software (E-level), the window drivers are almost as fast as with Lightning.

Unfortunately, in the Extended MODE4, the parameters of CON and SCR devices are not recognised by early versions of Qpac2, which will display them merely as SCR_ or CON_.

Also present as standard on Level-E drivers (and later) of the Emulators is the Pointer Environment, Toolkit II, the OUTLN command, a RAM disk driver and the Hotkey System II.

### 38.5.5 Microdrives

The emulator cannot support microdrives and if you try to access the microdrive, error -7 (not found) will be reported. If a program has been written for microdrives, either use

```
EXCHG flp1_file,'mdv','flp'
```

or

```
FLP_USE 'mdv'.
```

### 38.5.6 BEEP

The emulator cannot support QL sound and therefore this command usually has no effect.

### 38.5.7 MODE 8

This is not supported (except on some versions of the original QL-Emulator). Any attempt to access MODE 8 will have no effect, and displays in MODE 8 have the same effect as trying to load a MODE 8 screen in MODE 4.

All software will however run happily on the emulator (although see below if you are trying to use the Extended resolutions), although it will look a little odd.

### 38.5.8 MODE 4

Current versions of the emulator support a much enhanced screen resolution. This is known as Extended Resolution and on the Extended Mode 4 Emulator is chosen from the start-up screen (see above). On QVME, you can configure the size of the screen resolution or even alter it whilst the Emulator is being used.

This extended resolution mode has the same four colours as normal MODE 4, except that instead of displaying 512x256 pixels, the resolution of the screen is 768x280 pixels on the QL-Emulator EXTENDED4 and anything between 512x256 pixels and 1024x1024 pixels on the QVME (this can be any value in the range in steps of 8 or 16 pixels, provided that you have a powerful enough monitor).

Well written software must therefore not assume the resolution of the screen, and if writers wish to access these higher resolutions, the functions QFLIM, XLIM and YLIM have to be used. The logical consequence is that higher resolutions can only be supported with the help of the Pointer Environment, thus underlining that this extension is absolutely obligatory.

Unfortunately some software writes directly to the screen and assumes that the screen will be 512x256 pixels and start at address $20000. This will cause untold havoc in Extended MODE 4, although such software will run happily in normal MODE 4 on the emulator. Interestingly, this odd kind of software runs happily on QVME, because this has its own screen memory on-board and leaves the 32k RAM

from $20000 upwards untouched; so it does no harm if software writes directly into memory... you will simply not see the effect of this.

### 38.5.9 ROM Memory

The QL ROM on the emulator is actually stored in RAM, which means that if software tries to write to addresses in the range 0...65535, the Emulator is likely to crash. On a standard QL, writing to ROM has no effect. This should be avoided in all cases!

You can plug QL-ROM cartridges into the Atari ST with the help of special hardware.

### 38.5.10 Network

The Emulator cannot access the QL Network which was always very particular to Sinclair. This is really a pity. There does however now exist a means of communicating via the MIDI port to other STs (the MIDINET driver) and even the serial ports (the SERNET driver). See the separate Appendix concerning Networking.

### 38.5.11 Devices

The following devices are supported on the ST/QL emulators: flp, win, ram, dev, ser, par, prt, nulf, nulz, null, nulp, pipe_<length> pipe, pipe_name/pipe_name_<length>, sdump; where flp, win and ram are at 'Level-2'.

### 38.5.12 Lightning/Speedscreen

Neither Lightning nor Speedscreen can be used with current versions of the Emulator. Lightning could be used with drivers before Level-E, but you needed a special Atari version.

### 38.5.13 Qliberator

You will need to use v3.22a of the Runtimes at least on these Emulators.

## 38.6 A3.4.2 SMSQ/E

This is a commercial software emulator which can run on all ST models (but not the Falcon). It is fast and very flexible - in fact it is the operating system now sold with QVME, Extended4, and QPC emulators. This is available from Q-Branch, or from Jochen Merz Software. Note that there are several ST versions, call the supplier before ordering.

SMSQ/E is to be the new standard operating system for future QL developments and is also available for QXL II, AURORA, Q40 and QLs with either a Gold Card or Super Gold Card attached.

### 38.6.1 Lightning/Speedscreen

Neither Lightning nor Speedscreen can be used with SMSQ/E.

### 38.6.2 Qliberator

You will need to use v3.22a of the Runtimes at least on SMSQ/E.

Please refer to the SMSQ/E Appendix for more details.

**NOTE:**

The emulator is RAM based and you can therefore expect some problems with software which tries to write to the original QL ROM (in the range 0…65535).

## 38.7 A3.4.3 SMS2

This is a board which plugs into the side of the Atari ST computers, which was marketed by Furst Ltd. It is no longer available. SMS2 was not marketed as being an emulator for the QL, but as an add-on enhancement for the Atari's native operating system.

It includes a version of the Pointer Environment including QPac 2, and can run a fair amount of QL software. The main problem with SMS2 is that it does not provide a version of SuperBASIC, although it is possible to create programs under SMS2 using the in-built version of the QD editor (© Jochen Merz Software) and the in-built Qliberator compiler (© Liberation Software).

Unfortunately, the use of this board is restricted, since it only worked on Atari ST computers. It would also work on Atari STE computers however, provided that the QVME board was plugged in also!

The way in which SMS2 loads programs is very different to other implementations of SuperBASIC due to the lack of an interpreter. We feel that this is beyond the scope of this book.

SMS2 provides the following facilities as well as being able to run various QL software:

- Access to Atari serial and parallel ports (details of ports unknown)
- Access to Atari floppy disks and SCSI hard drives (presumably it can handle QL disks)
- Network facilities are available for SMS2 via the MIDINET extension (now provided with SMSQ/E).
- Built in ram disks
- Supports Atari mouse and Atari monochrome display (640x400 pixels)

## 38.8 A3.5 Commodore Amigas

### 38.8.1 A3.5.1 Amiga QDOS

This is a public domain software emulator available for Amiga computers. It was distributed together with a load of public domain QL software on a CD cover disk on the Amiga Format magazine (published by Future Publishing of Bath) in September 1996. It is also available from Qubbesoft P/D.

Details about the emulator are on the Web at:

http://www.emulnews.com/aer/articles/af (This link no longer works. NDunbar)

The program loads the operating system from disk and basically simulates a JS ROM QL with a few additions in later versions. Although a public domain toolkit is included with the package that contains many of the commands added by Toolkit II, you really could do with a copy of Toolkit II on disk to load into the Emulator (with LBYTES flp1_Toolkit2_cde,49152).

There is no need for the EPROM_LOAD command on this Emulator since, once any toolkits have been loaded into the Amiga's memory (as with Toolkit II above), you can do a warm reset of the system by pressing <CTRL><SHIFT><ALT><TAB> which will not wipe out any code previously loaded into the QL's EPROM area.

The emulator has been (and is still being) improved independently by several people; making it impossible to be certain of which versions have which bugs in them.

It is recommended that you get at least v3.23 which had the following enhancements over earlier versions:

- Supports the full range of Motorola processors (68000, 68010, 68020, 68030, 68040 and 68060).
- MODE 8 support (excluding FLASH)
- Authentic BEEP sounds which are the same as on the original QL.
- QL compatible disk handling, including the ability to use QL HD disks and sub-directories.
- The system variable SYS_PTYP (at offset $A1) is supported, allowing you to test the type of processor on which Amiga QDOS is running.
- Support for dual screen display MODE.

This emulator is also available in the form of QDOS Classic, which has been released for use on the Q40.

**NOTES:**

Memory The emulator is RAM based and you can therefore expect some problems with software which tries to write to the original QL ROM (in the range 0…65535).

### 38.8.2 ROM Cartridges

These cannot be connected to the Amiga.

### 38.8.3 Network & Microdrives

As with the ST/QL Emulator, none of these are supported.

### 38.8.4 MODE 8

Before v3.23 this was not supported and any attempt to use this will result in MODE 4.

### 38.8.5 BEEP

Before v3.23 this was not supported.

### 38.8.6 MODE 4

Because of the way in which the Amiga's display works, some displays can cause flickering of the screen, or even a scrolling screen. This has to be controlled by altering the speed at which the Amiga's Blitter chip updates the screen (on early versions of the emulator, this was achieved by using POKE 164082,x). In v3.20+ SCR_PRIORITY was added to perform this task.

You can actually alter the four colours available in MODE 4 if you wish, by POKEing the hardware. To do this, you will need to POKE_W a new word value (up to 4095) into one of the following addresses, each one representing one of the QL's standard 4 colours (note the need for quote marks around the address due to the limitations of QL maths):

```
POKE_W '14676352', black
POKE_W '14676354', red
POKE_W '14676356', green
POKE_W '14676358', white
```

Beware that you should not try to read the values at these addresses (for example with PEEK_W) as this is likely to alter the contents!

### 38.8.7 DEVICES

The standard QL devices (except MDV) are all supported without any alterations. However, the Qjump static RAM disk supplied as RAMPRT does not work. Unfortunately, disk access is somewhat slower than on the QL in all current versions of the emulator.

In v3.10 the serial port could successfully receive data from the QL at up to 9600 BAUD. v3.20 managed to send data to the QL at up to 1200 BAUD and to the Apple Macintosh at up to 19200 BAUD. We do not know at present whether later versions have improved these figures.

Disk formatting was also exceptionally slow prior to v3.23 and before v3.10, if Amiga-QDOS wrote to a disk, it could not be read on a standard QL.

### 38.8.8 TAS INSTRUCTIONS

The main area of incompatibility with the Amiga QL Emulator is the fact that the machine code TAS instruction, which is used to test and set a byte in one command, does not work properly on the Amiga. However, there is a small program supplied with the emulator which patches these instructions in any given program. All programs compiled with Turbo and SuperCHARGE need to be altered in this way. If a program has been compiled with Qliberator, you will need to patch the runtimes in this way.

Note that this incompatibility has been completely cured on certain versions.

## 38.9 A3.6 Unix Systems

### 38.9.1 A3.6.1 UQLX

This is a shareware software emulator by Richard Zidlicky, still in an early development stage.

In order to work it requires Unix or a Unix-like operating system plus both gcc and Xwindows. It will however work on at least 5 types of processor: HP-PA, INTEL (486 or better), MIPS, PPC and SPARC. If you use Linux on the Q40, this emulator can be used as another method to allow the Q40 to boot up as either a Linux machine or a QL!!

Current versions support JS ROMs (or Minerva), Toolkit II and MODE 4 displays. You can access the floppy disks and up to 4MB of RAM. You can also create and access UNIX directories using Level-2 Device Drivers.

If you use Minerva, you can use higher resolution display modes (up to 8192x4096 pixels) and access 16MB of RAM.

The main incompatibility problem with this emulator is due to the case sensitive names used by Unix (ie. filenames).

CHAPTER

THIRTYNINE

# A4 THOR COMPUTERS

## 39.1 A4.1 Introduction

Although we do not have direct access to a THOR XVI computer, we are able to report on one or two comments passed onto us by users.

The THOR XVI was the last in a line of various QL compatible computers originally produced by CST in the United Kingdom, and more latterly produced in Denmark by Dansoft.

This computer was a very nice package, with much improved hardware and also improvements to the operating system ARGOS. Unfortunately, not many of these computers seem to have been produced and the constant changes to ARGOS have made it very difficult to know where the current problems lie. There now seems little prospect of this computer ever taking off again, and it is highly unlikely that programs will be altered to suit this computer.

v6.40 of the THOR ROM contained various disastrous bugs and it is recommended that users alter their ROMs to either v6.39 or v6.41 (the last ROM version to be produced). However, with the apparent demise of Dansoft, you will need to try and contact other THOR members to see if they will let you use their ROMs to re-blow your chips.

ARGOS is generally based on an MG ROM with parts of Toolkit II added on - this helps to explain why only some of the THOR's commands support default directories. Although some of the bugs are fixed, many are left well alone, and (at least on some versions of ARGOS) new ones are introduced, which can make some programs unreliable on the THOR. We have attempted to highlight many of the known problems in the main body of this manual, but this section contains one or two further sticking points.

## 39.2 A4.2 KEYROW

The main problem with the THOR range of computers has always been their handling of the KEYROW function, which is not supported on all of the range. This was partially implemented on the THOR XVI to try and retain compatibility but programs which use KEYROW cannot be guaranteed to run on the THOR.

## 39.3 A4.3 MODE

The THOR XVI introduced a new MODE 12 which provides the same resolution as MODE 8 (256x256 pixels), but replaces the FLASH bit by an intensity bit which enables the THOR XVI to display 16 colours on screen.

## 39.4 A4.4 The Thor Windowing System

There has never been a version of the Pointer Environment which will run happily on the THOR XVI - instead Dansoft produced its own windowing environment which moves the system variables and the start address of the screen with all of the inherent problems faced by Minerva users.

This can however be turned off on start-up by using:

```
CLOSE
POKE SYS_VARS+133,-1
OPEN #1,con_
OPEN #2,con_
OPEN #0,con_
WMON 4
```

## 39.5 A4.5 BEEP

Although implemented on the THOR XVI, this is unlikely to match with the sounds produced on the QL.

CHAPTER

FORTY

# A5 EXPANSION BOARDS

In this section, we give a brief description of the more common expansion boards which are available for the standard QL which go further than merely increasing the memory of the machine or adding the ability to read disk drives (or hard disks), normally improving the speed of the machine and providing additional toolkit facilities.

## 40.1 A5.1 GOLD CARD

This expansion card, unlike other memory and disk expansion cards for the Sinclair QL provides a massive increase of memory (a total of 1920K) for the QL as well as increasing the speed of the QL by about 4 times.

Gold Card also provides Level-2 device drivers as standard and allows the QL to access up to three Double Density (DD), High Density (HD) or Extra-High Density (ED) disk drives. Toolkit II and a battery backed clock is also in-built.

The main problem which may be encountered by users and software authors alike, is that it copies the QL's ROM into RAM, which means that like the emulators, any programs which try to write to addresses in the range 0..65535 are likely to crash the computer.

It should also be noted that some combinations of Gold Cards and QLs allow POKE 114796,0 to force the Gold Card to run at 24MHz (it normally runs at 16MHz compared to the QL's 12MHz). The circumstances where this works are undefined at the moment, but see below. Some users have however reported problems with some software running at this speed, and in particular the network and microdrives are unusable. The Gold Card can be returned to normal with POKE 114796,255.

The so-called Gold Card Go-Faster POKE works on a certain series of Gold Cards, in connection with random effects only. We highly dis-recommend the use of this POKE, it may overheat the processor and damage your Gold Card. Please, forget about it.

Gold Cards will support the SMSQ/E operating system if you wish to use this. A Gold Card is the minimum expansion board required to run an AURORA.

## 40.2 A5.2 SUPER GOLD CARD

This board provides the same facilities as the Gold Card plus much more. It is much faster, being approximately 3 times quicker than a Gold Card. It uses a 68020 chip and provides 3986K of memory.

It suffers from the same problem as the Gold Card in that it copies the operating system into RAM.

This card also allows you to set a flag (using AUTO_TK2F1, AUTO_TK2F2 and AUTO_DIS) to say whether the QL should automatically start up in either Monitor or TV mode and with or without Toolkit II present.

Other additions include an in-built parallel port and the ability to use four disk drives. There is also a cache on- board the processor which can make programs run much quicker (although you may need to use CACHE_OFF to disable this to make some programs work).

You can also automatically disable or enable the second screen (see SCR2DIS) which is provided by Minerva and also used by some software - this can therefore cause problems with some programs, especially games.

Again, this card can be used to with SMSQ/E and AURORA.

## 40.3 A5.3 AURORA

AURORA allows the QL to display much higher resolutions on Monitors (up to 1024x768 pixels) and also speeds up the operation of the QL. It is no longer available as new, but may be obtained second hand. Aurora is a replacement for the QL motherboard, and needs a Gold Card or Super Gold Card to work, together with various chips from the original QL board (including an operating system).

You would normally use this board to build a replacement QL in a PC tower case (although it is just about possible to use the original QL case). You will also need some floppy disk drives (the microdrives are not supported), a Qplane (to let you plug the various boards together), a keyboard interface, an SVGA or QL monitor and cables. The BraQuet from Q Branch and the MPlane from TF Services are also recommended to ease the assembly of the system (the latter is a substitute for Qplane).

If you intend using Minerva with AURORA, you will need at least v1.86. It is also recommended that you use AURORA with a Super Gold Card as the standard Gold Card limits what you can do with the enhanced graphics capabilities.

Although any QL operating system can be used with AURORA, you will need SMSQ/E to make use of the higher graphics resolutions.

Several programs have difficulty working under Aurora and SMSQ/E v2.75+ (these later versions allow the use of the higher resolution screens). The reason for this is that even if SMSQ/E is configured to start in 512x256 resolution mode, Aurora uses a fixed screen width of 256 bytes (instead of the normal 128 bytes in this mode) and therefore has to also move the screen from the normal base address.

To try and overcome this problem, Aurora copies anything written to the old screen address (131072) across to the top left hand corner of the new screen. However, this is not a two-way process and therefore if Aurora's screen is altered (for example by BASIC PRINT commands), this alteration will not appear on the picture stored at the old screen address, making a mockery of hand-written machine code pan / scroll routines for example.

Luckily, anyone using Aurora will have access to a second operating system which can be used to run the programs successfully. After all, it is this second operating system which is used to load in SMSQ/E!!

## 40.4 A5.4 Q40

Q40 is a replacement motherboard for the QL. It is supplied with 16Mb RAM (although it will support up to 32Mb) and an I/O card. You will need to add a tower case, a keyboard, floppy and hard disk drives, a standard PC monitor and mouse. The Q40 even has the ability to drive stereo sound, using an in-built digital to analogue converter which can be used with either speakers or headphones. All drivers and equipment to connect these items to the motherboard are built in.

The Q40 is a full-blown 68040 processor at 40Mhz with a maths co-processor. This makes it an extremely quick version of the QL. It is hoped that later versions with a 68060 processor will be available in due course.

You can use either Linux, SMSQ/E or Classic QDOS (developed from the Amiga QDOS Emulator) as the main operating system with the board. These will all make the most of the Motorola processor, making the Q40 a good competitor to a standard PC. There is also the added benefit that with Linux working on the machine, the Q40 has the ability to access the internet.

Although Q40 supports the original QL screen (at base 131072), it also allows the QL to display much higher resolutions on Monitors (up to 1024x768 pixels).

Further, there is the added bonus that with SMSQ/E, the Q40 now supports enhanced colour modes, including a 256 colour mode and a 24 bit full colour mode, which can vary fom program to program. Background wallpapers can also be loaded, meaning that this is a very flexible QL successor indeed.

The Q40 is available from QBranch.

## 40.5 A5.5 HERMES / SuperHERMES

Hermes is a replacement board for one of the QL's microchips (the 8049), which is also used on the AURORA replacement mother board - Hermes is available from TF Services. Hermes provides the QL with much more reliable serial port communications, improved sound and keyboard.

Hermes fixes some of the problems inherent in the original QL, including supporting independent input BAUD rates for each serial port (not necessarily the same as the output BAUD rates), together with full support for input BAUD rates of 19200. Problems with the QL's BEEP command are also fixed, meaning the pitch of the sound does not affect its duration.

SuperHERMES is an improved version which also adds an additional high speed serial port (up to 57600 bps), three low speed serial ports (30 bps to 1200 bps) which can be used for a serial mouse or a graphics tablet. This also includes a keyboard interface (to allow you to use a full size serial IBM AT keyboard), a capslock/scrollock LED connector and 1.5K of RAM which can store data whilst the QL is switched off. Unfortunately at present, without a special public domain program written specifically for use with SMSQ/E, you cannot use a keyboard linked to SuperHERMES to reset the computer (for example with the soft-reset provided with Minerva or SMSQ/E).

If you want to use independent BAUD rates, you will need SMSQ/E or Minerva as well as Hermes/SuperHERMES.

## 40.6 A5.6 QuBIDE

QuBIDE is a board which provides the QL with access to modern PC Hard Disks and is no longer available new. If you intend using this board, you will need expanded memory (expansion cards can be plugged into the QuBIDE interface).

Early versions of QuBIDE will work with most IDE standard Hard Disks and contain WIN_DRIVE, WIN_USE and MAKE_DIR commands similar to those listed in the manual. However, the MAKE_DIR command will not work if any files already exist which would be inside the sub-directory (unlike the standard implementation of this command).

There is also a v2.xx ROM available for QuBIDE, which allows access to a wider range of Hard Disks (ATAPI/IDE standard). You can also specify that MAKE_DIR will work as per the standard implementation. A trashcan facility is also added, where deleted files are simply moved to the 'trashcan' and have to be specifically removed at a later date (similar to the Recycle Bin on Windows 95).

Unfortunately, problems have been reported with the Trashcan facility which makes it unreliable.

CHAPTER

FORTYONE

# A6 COMPATIBILITY

Most QL software is well written, and provided that you use standard SuperBASIC commands and make no assumptions such as about addresses in memory, or the size and location of the screen, it would appear to work happily on all different QL and QDOS compatible set-ups. Unfortunately, as always, there are some exceptions to this rule, and the areas which appear in the following sections would appear to cause the greatest problems:

## 41.1 A6.1 Addressing

As the QL was developed, the designers tried to leave everything open-ended so that nothing could be taken for granted. However, towards the end of 1984 (the first year of the QL's long history) things appeared to have settled down with the JS ROM in the UK and USA, and MG ROM version elsewhere. Without any later ROMs in sight, software writers got very lazy and rather than write a few lines of machine code to check for the address for various things (such as the start of the screen): they assumed that it would always remain where it had been, and so started to use absolute addressing.

Likewise, programmers assumed that the QL's operating system would always appear in ROM and they could therefore write routines which tried to overwrite parts of memory, regardless of whether they were pointing to ROM or RAM - after all, it could not harm the operating system as you cannot write to ROM, can you (!).

Another problem are programs which use RESPR to reserve memory for themselves at a particular place - the boot programs typically include lines such as:

```
10 Addr=273102
20 A=RESPR(0):A=RESPR(A-Addr)
30 LBYTES flp1_Program_cde,Addr:CALL Addr
```

Unfortunately, later developments in the QL world, namely emulators and the Gold Card (for example), have moved the QL's operating system into RAM, meaning that it can be overwritten, thus crashing the whole machine. Parts of memory are moved around freely, making use of the calls incorporated into QDOS by the QL's designers and even the speed of the QL has altered. Despite the best attempts of the manufacturers of the Gold Card and emulators, this has resulted in one or two incontestabilities with older software.

## 41.2 A6.2 Speed

Both QDOS emulators and the Gold Card have greatly increased the speed at which the QL works, making some programs unusable. Luckily the effects of this are limited by various commands which slow the operating speed of emulators and the Gold Card down.

## 41.3 A6.3 The Operating System

Some software in the market was written with specific versions of the QL ROM in mind. For example, one program which was quite a useful SuperBASIC utility, would appear to only work on JS and MG ROMs. This software can only become redundant as more and more users upgrade their systems to take advantage of the latest developments in the QDOS operating system.

## 41.4 A6.4 Memory

Some older software is not address independent, which means that it has to be loaded at a specific place in memory. This can prove impossible on machines with expanded memory, but luckily commands do exist to reduce the memory size. We have however come across one program, which although it is address independent, refuses to work correctly whenever the system on which it is running has anything more than the QL's original 128K (even if only 128K is set aside for use by the program).

## 41.5 A6.5 The Stack Pointer

As any machine code programmer will be aware, the processor's address register a7 is used by the operating system as the stack pointer. Some software attempts to set this to an absolute address when the program begins (even though there is no need for this). If such a program is not executed as a task, then it is likely to fall over on Minerva and SMSQ/E.

## 41.6 A6.6 Compilers

SuperBASIC compilers are an excellent means of getting the best of two worlds: the flexibility and clarity of a SuperBASIC program, but with the speed of machine code. Unfortunately, although the SuperBASIC compilers have mainly kept pace with the development of QDOS, some software originally compiled with earlier versions of compilers has not been upgraded, meaning that it may not be compatible with the latest ROM versions.

There are two types of SuperBASIC compiler which have been produced for the QL, namely true compilers (Turbo and Supercharge) which produce independent machine code, not relying in any way on SuperBASIC structures (and the code produced is therefore much more portable between different QDOS machines) and so-called pseudo compilers (Qliberator) which produce machine code which still uses SuperBASIC structures and calls.

Although the former produce much quicker code, they are not as versatile as the latter as they expect SuperBASIC commands to be used in a certain manner and cannot therefore recognise the enhancements introduced to SuperBASIC commands by Minerva SMSQ/E and emulators.

There is also a problem in that programs compiled with Turbo will not work on versions 2.25 - 2.31 of SMSQ. You will also find that some Turbo compiled programs will not work if started from a copy of BASIC other than Job 0 (on both SMS and Minerva) and also neither compiler can currently compile a program which from within a multiple copy of BASIC.

Patch programs have been released which enable Turbo compiled programs to be used on a system which has a large amount of memory (such as the Q40). Later versions of the Turbo Toolkit (v3d27 and later) are also required for better compatibility.

However, one of the remaining problems with SMSQ/E which is displayed by Turbo compiled programs is the failure of the in- built Integer to ASCII conversion routine to cope with negative integers. This means that Turbo compiled programs remain unreliable on current versions of SMSQ/E - refer to PRINT for an example.

## 41.7 A6.7 High Resolution Displays

More and more QL implementations are now able to use much higher screen resolutions than the original 512x256 pixels (or 256x256 pixels in MODE 8). Unfortunately many programs were written before this facility was available and therefore will not work correctly on higher resolution displays. Even if they do work, the program may be confined to a small section of the screen, normally the top left hand corner, and use fonts which are much too small to read.

The only answer to this (unless a new version of the program is released) is to use a lower resolution screen (see DISP_SIZE) or to put up with the slight inconvenience. Aurora has other problems too - see Appendix 5.3 for more details.

You should however be aware that on some monitors, a lower resolution screen may still not fill up the whole area shown by the monitor - this does not cause a problem in itself and there is nothing you can do about it - it is due to the difference in ratio between 512x256 pixel displays and standard PC (or ATARI / MAC / UNIX . . . ) displays.

Appendix 16.4 is also of interest in connection with High Resolution Displays.

## 41.8 A6.8 String Lengths

The maximum length of strings varies on each QL implementation (even though you can use DIM to dimension a string up to 32767 characters, this does not mean that you will be able to use all of those characters!!). SMSQ/E allows a maximum string length of 32765 characters, whilst Minerva allows a maximum string length of 32764. QDOS ROMs allow a maximum of 32766 characters. On the other hand FILL$ is allowed to be used to create slightly different string lengths - on SMSQ/E this is 32764 characters, on Minerva 32767 (except on v1.98 where a limit of 32764 characters was implemented) and on QDOS ROMs, FILL$ can produce strings up to 32767 characters long.

The outcome of this is that the maximum length that should be used for a string should be 32764 characters which is the limit imposed by the Turbo compiler.

## 41.9 A6.9 Later Processors & Gold Cards

Various QL implementations use a chip as the main processor which is not a 68008 chip (the chip that the QL was originally designed to use). These later chips have various facilities, such as caches which can cause problems with some software (see CACHE_OFF). You may also note that some leisure software does not work on a GOLD CARD QL - this is normally where the command SCR2DIS has been issued - some leisure software insists that the second screen is enabled!!

## 41.10 A6.10 Finally

Specific points to watch out for on the different QDOS implementations are covered in the appendices dealing with each one.

There are also various points explained in the main keywords section of this book.

CHAPTER

FORTYTWO

# A7 MULTIPLE BASICS

The Sinclair QL is one of a few computers which allow you to run several BASIC programs in memory at the same time, multitasking them as if they were machine code programs.

The QL implementation of this is more flexible than most other implementations and certainly a lot cheaper, requiring only a standard QL with Minerva ROM (or a QL with Gold Card and SMSQ/E) at the least.

Both Minerva and SMS provide the ability to use several SuperBASIC interpreters at the same time, allowing you to work on more than one BASIC program at a time and run them alongside other BASIC programs.

There is not really a great deal of difference between the two implementations and so we shall first of all describe the Minerva implementation (known as MultiBASICs) and then describe the differences in SMS's implementation (known as Multiple SBASICs).

## 42.1 A7.1 MINERVA MultiBASICS

A MultiBasic on Minerva is very similar to the standard QL's SuperBasic interpreter. This means that once a MultiBasic has been created, you can use it in practically the same way as you would normal SuperBasic (that is to say that you can enter programs, load and run programs using the standard commands set out in this manual).

There are several advantages in using MultiBasics, but the main advantages are:

- You can have several programs running at the same time (one program under each MultiBasic, and one under the standard SuperBasic interpreter) in much the same way as you can have several machine code programs
- You can tell a MultiBasic to ignore any machine code extensions other than the standard ROM keywords (letting you test programs on semi-clean machines)
- If a program 'locks up the QL', provided that it is running in a MultiBasic, it will only lock up that interpreter and you should be able to return to the standard SuperBasic interpreter.

A Multibasic can be created in one of three ways:

EXEC 'flp1_multib_exe' *[,file^x]* [;cmd$]

or:

MB

or:

EXEC pipep *[,file^x]* [; cmd$]

The first two methods are for use on Minerva ROMs pre v1.93 (the file multib_exe is contained on the disk supplied with the Minerva ROM). The second method is for use on later versions of Minerva which have the relevant machine code built into the ROM.

### 42.1.1 A7.1.1 Channels OPENed automatically in MultiBASICs

When one of the above commands is entered, Minerva will examine the parameters passed using the command. It first of all has to decide how to set up the standard channels (#0 - the primary command channel and #1 - the primary output channel).

If neither a file nor cmd$ is supplied, for example:

```
EXEC pipep
```

then a single window is opened on screen which is used by both #0 and #1. #2 is not opened. The actual position of this window on the screen cannot be set by the user and is dictated by how many MultiBasics are already running (you can of course redefine #0 and #1 from within the MultiBasic by using WINDOW, although as both #0 and #1 use the same window, any attempt to redefine #0 will affect #1 and vice versa).

If however, cmd$ is supplied, for example:

```
EXEC pipep;'This is a Command String'
```

channels #0, #1 and #2 will all be set up as in the standard SuperBasic interpreter. You can then CLOSE #1 or CLOSE #2 without removing the MultiBasic. Sections A7.1.2 to A7.1.4 explain how the command string is dealt with.

If one file (or channel) is supplied, for example:

```
EXEC pipep,flp1_inputfile
```

Minerva will open both #0 and #1 to access that file, whereas if two files are supplied, #0 is opened to access file1 and #1 is opened to access file2. If three or more files are supplied, then #0 is opened to access file1, #1 to access file2, #3 to access file3 and so on (#2 is omitted).

### 42.1.2 A7.1.2 Starting a MultiBASIC with the Original QL ROM Commands only

cmd$ is used to pass different parameters to a MultiBasic. If the last character is an exclamation mark (!) then this is taken to be the ROM marker, and the MultiBasic will start up recognising only the original keywords contained in the Minerva ROM.

### 42.1.3 A7.1.3 Multitasking a MultiBASIC Program

In order to run a BASIC program as a multitasking program, it is necessary to start a MultiBASIC interpreter and pass the name of the program to be run as part of cmd$ passed to the MultiBASIC by the start-up command.

If cmd$ contains the file marker (>), the characters in the string before that marker are taken to represent a file name which will be opened to read commands from (similar to a stream device). The MultiBasic will open both #0 and #1 to this filename and will read the characters from the file and try to interpret them as a program.

If the program does not open any screen windows and print to them, this will allow a SuperBasic program to run in the background, for example, as a filter.

You will be unable to redefine #0 from within the program, as this will stop the MultiBasic from accessing the command file. This could for example, be used to test programs:

```
EXEC pipep;'flp1_boot>'
```

### 42.1.4 A7.1.4 What Happens to the Rest of the Command String?

Having stripped all of the information needed in Section A7.1.2 and A7.1.3 from cmd$, any characters left in the string supplied can then be read from within a MultiBasic itself by simply accessing CMD$. For example:

```
PRINT 'Your Name is :'!CMD$
```

### 42.1.5 A7.1.5 Loading Toolkits into a MultiBASIC

Any toolkits which are LRESPR'd from within a MultiBASIC are defined as local to that MultiBASIC and will not be recognised from any other interpreter (unless you start yet another MultiBASIC from within that MultiBASIC interpreter). They will therefore be removed when you remove the MultiBASIC interpreter which loaded the extensions.

For this reason, MultiBASICs should not be used to link in system extensions (such as BTOOL which adds new device drivers).

## 42.2 A7.2 SMS Multiple SBASICs

These are extremely similar to Minerva MultiBASICs and can be used in much the same way as MultiBASICs and have the same advantages (except that you cannot at present tell SMS to start up an SBASIC with only the standard QL ROM keywords).

On versions of SMSQ which do not incorporate the Hotkey System II automatically (and therefore need the file HOT_REXT to be loaded), you will need to enter the command SB_THING to create the SBASIC Thing (see below).

A Multiple SBASIC can be created in one of several ways. The more usual methods are:

EXEC 'flp1_program_bas' *[,file^x]* [;cmd$]

or:

SBASIC [offset]

or:

EXEP 'SBASIC';cmd$

The first method allows you to automatically load a BASIC program to run under an SBASIC interpreter multitasking alongside the normal SuperBASIC interpreter (Job 0). This allows you to start a program up from within Qpac 2's File Menu.

The second method merely starts up an SBASIC interpreter.

Because SBASIC is implemented as a Thing under the Hotkey System II, you can also set start an SBASIC Interpreter using the third method, or even from Qpac 2's Exec Menu. You can even set up a hotkey to start an SBASIC interpreter. For example:

```
ERT HOT_THING ('L','SBASIC')
```

will start up a new SBASIC interpreter whenever <ALT><L> is pressed.

### 42.2.1 A7.2.1 Channels OPENed automatically in SBASICs

If you use the command SBASIC to start an Interpreter, the initial windows which will be OPEN depends upon whether an offset parameter is passed:

- If no offset is passed then all the standard Windows #0,#1 and #2 will be OPENed (as per WMON)
- If offset is specified, only #0 will be opened and the offset is used to determine the location on screen of that window.

If you use the third method of invoking SBASIC, or Qpac 2's Exec Menu or a Hot Key to start an Interpreter, then it depends upon whether you pass a string as a parameter:

- If no string is passed, then the standard windows #0, #1 and #2 are OPENed.
- If you pass a string to the interpreter, then no windows are OPENed and the string is treated as a command as if it had been entered in the command line (see Section A7.2.7 below!). For example:

```
EXEP 'SBASIC';'LRUN flp1_PROG_Bas'
```

is the same as:

```
EXEC flp1_PROG_bas
```

If you instead use a command such as EXEC to start up a program under a SBASIC interpreter, then no windows will be OPENed by default and the program will need to OPEN all of its own channels.

However, if any files (or open channels) are specified then (as with MultiBASICs) these are OPENed as #0, #1 onwards (#2 is not omitted in this case).

We would refer you to the explanation of EW about setting up Filters by making use of these facilities.

### 42.2.2 A7.2.2 The Command String

The effect of the command string depends upon the circumstances.

If the SBASIC Interpreter is started using the EXEC command (or similar) then SMS does nothing with the command string and it can merely be read from within the SBASIC Interpreter with the function CMD$.

If however, SBASIC is started using the Thing System, then the command string is executed as if it were a direct command (see Section A7.2.1).

### 42.2.3 A7.2.3 Starting an SBASIC with the Original QL ROM Commands only

This is currently not possible.

### 42.2.4 A7.2.4 Multitasking an SBASIC Program

This is much easier than under MultiBASIC and the standard form for doing this is to use a command such as:

```
EXEC flp1_program_bas
```

Provided that the name of the program ends in _bas or _sav, then this BASIC program will be started as a separate multitasking program running under an SBASIC Interpreter.

You can also use SBASIC's characteristics as a Thing to start a BASIC program - see Section A7.2.1.

### 42.2.5 A7.2.5 Loading Toolkits into an SBASIC

This follows exactly the same rules as on a MultiBASIC.

### 42.2.6 A7.2.6 Defining the Name of an SBASIC

You can do this by using the command JOB_NAME from within the SBASIC Interpreter.

If you start an SBASIC using the HOT_THING command, you can also use this to define the name of the Job, for example:

```
ERT HOT_THING('L','SBASIC','INT 1')
```

However, all future SBASICs started from the hot key will still be given the same name!!

### 42.2.7 A7.2.7 Channel #0, #1 and #2

Channel #0, #1 and #2 are dealt with differently under an SBASIC Interpreter than under the main SuperBASIC Interpreter (due to the fact that they may not be OPEN - see Section A7.2.1).

- All standard QL ROM and Toolkit II commands which would normally default to #1 or #2 will access #0 if the relevant default channel is not OPEN.
- If a standard keyword tries to access #0 by default (or as in the previous paragraph), if #0 is not OPEN already, then a small default #0 will be OPENed automatically.
- If you have RUN a program under SBASIC (for example used EXEC flp1_PROG_BAS) and on completion of the program #0 is not OPEN, the SBASIC Interpreter will be removed.
- If an error occurs and #0 is not OPEN, a small default #0 will be OPENed automatically to report the error.

### 42.2.8 A7.2.8 Removing an SBASIC

CLOSE #0 will remove an SBASIC if a program is not RUNning. However, it is better to use the explicit command QUIT.

### 42.2.9 A7.2.9 Keywords Which are Useful in SBASICs

Reference should be made to SEND_EVENT and WAIT_EVENT.

DEVTYPE allows you to find out if a channel is OPEN.

QUIT and JOB_NAME are only of any relevance from within an SBASIC.

WMON and WTV allow you to move the SBASIC windows.

CHAPTER

FORTYTHREE

# A8 ERROR MESSAGES

The QDOS error code is a negative integer between -1 and -21 and is often referred to instead of the relative error message. You can either refer to the following tables to find the message text or use the various toolkit implementations of the command REPORT to display the representation on your machine.

On some implementations, REPORT also accepts parameters between -22 and -27 which represent strings used by the operating system, but which are not strictly error messages. The text can be freely changed with the TRA and LANG_USE commands.

The instances where such error messages will be reported is actually dependent upon the task which was being performed at the time.

## 43.1 A8.1 Standard English Error Messages

A very general explanation of each message is given below, according to the error number. For each message both the original English QDOS error message is given as well as the new English SMS error message:

**-1 NOT COMPLETE (QDOS)-INCOMPLETE (SMS)**

This message is generally issued when the Break key <CTRL><SPACE> is pressed, and signifies that a task being carried out by the QL has been interrupted.

This message will also appear if you try to use the standard version of RESPR when a Job is loaded into the QL.

**-2 INVALID JOB (QDOS)-INVALID JOB ID (SMS)**

This message is issued by all Job-related commands when the Job identification number / Job name / Job tag supplied as a parameter to the command does not relate to a Job resident in the QL's memory.

**-3 OUT OF MEMORY (QDOS)-INSUFFICIENT MEMORY (SMS)**

This message is quite self-explanatory. It is issued when you try to do something which requires more memory than is currently available. This can however be due to heap fragmentation, and it may therefore be useful to try the command DEL_DEFB.

**-4 OUT OF RANGE (QDOS)-VALUE OUT OF RANGE (SMS)**

This generally occurs when a parameter supplied to a machine code Function or Procedure cannot be handled by that machine code routine. The best example of this is trying to open a window which cannot fit on the screen, eg:

```
OPEN #3,SCR_10000x500a0x0
```

**-5 BUFFER FULL (QDOS & SMS)**

This error generally occurs if SuperBasic's input buffer becomes full. On pre-JS ROMs, you are likely to see this error quite often, especially if you try to INPUT a line greater than 128 characters.

**-6 CHANNEL NOT OPEN (QDOS) - INVALID CHANNEL ID (SMS)**

This will be generated by all well-written machine code Procedures and Functions if you pass a channel parameter (#ch) which points to a channel which is not actually open.

**-7 NOT FOUND (QDOS & SMS)**

This error message is generally issued by file-related commands (eg. LOAD) if either the supplied device name or file name do not exist.

**-8 ALREADY EXISTS (QDOS & SMS)**

This error is returned by commands such as SAVE to indicate that a file already exists with the specified name on the specified device. If Toolkit II is available, you will generally be asked whether or not you wish to overwrite the file.

**-9 IN USE (QDOS)-IS IN USE (SMS)**

This message is normally generated by file-related commands where you try to access a file which has an exclusive channel open to it, or for example, if you try to DELETE a file which has a channel open to it. This message is also printed by commands such as WSTAT where a channel is open to a file, allowing further data to be written to that file.

You will also see this message if you try to open a channel to one of the serial ports when there is already a channel open to that port.

**-10 END OF FILE (QDOS & SMS)**

You will see this message if you try to input data from a file when the file pointer is at the end of the file.

This also occurs if you try to READ DATA from within a program and there is no more DATA in the program to be READ. Note however, on SMS, that this error is altered to:

**End of DATA**

in this instance.

EOF or EOFW should be used to overcome this error.

**-11 DRIVE FULL (QDOS)-MEDIUM IS FULL (SMS)**

This error message is normally generated when you try to write to a medium and there is not enough room on the medium. Unfortunately, with many commands, unless you have Toolkit II installed, the error message will only be generated if there is no room left on the medium when you first try to open the new file. Without Toolkit II, if the medium becomes full whilst the file is actually being written, no error will be reported and an incomplete file will be left on the medium.

**-12 BAD NAME (QDOS)-INVALID NAME (SMS)**

This error is generated when you try to use an undefined name as a command in a program. It generally reveals typing errors in programs, such as 10 PRIT 'Title'.

**-13 XMIT ERROR (QDOS)-TRANSMISSION ERROR (SMS)**

This is generated when you are trying to read data over the Network, or serial ports. This error normally occurs when there is an error in the parity of the data which has been read.

**-14 FORMAT FAILED (QDOS & SMS)**

This error will appear when you try to FORMAT a medium. It generally shows that there is something dreadfully wrong with that medium, however, you may find that if you try to FORMAT the same medium in another drive, or clean the drive heads, this may prevent this error.

**-15 BAD PARAMETER (QDOS)-INVALID PARAMETER (SMS)**

This message is generated by machine code Procedures and Functions where the wrong type of parameter has been used in the calling statement. This may for example occur if you try to pass a string when a number is required.

**-16 BAD OR CHANGED MEDIUM(QDOS) - MEDIUM CHECK FAILED(SMS)**

This message occurs when you try to read or write to a medium and an error occurs when the computer tries to verify the data being read or written.

When trying to write to microdrives, it will also signify when a microdrive is read only, due to a bug in the QL's hardware.

**-17 ERROR IN EXPRESSION (QODS & SMS)**

This error is normally generated when part of an expression does not make sense, for example: DIM a(10,10):PRINT a/10. Look for the use of undefined variables, or possibly arrays where a simple variable is needed (or vice versa).

**-18 OVERFLOW (QDOS)-ARITHMETIC OVERFLOW (SMS)**

This occurs when you have used an expression which cannot be handled by the QL's maths package, for example divide by zero, or where you try to assign a value to an integer which is greater than 32767.

**-19 NOT IMPLEMENTED YET (QDOS) - NOT IMPLEMENTED (SMS)**

This message generally appears when you try to do something which the QL cannot currently do, but which it is hoped may be implemented in the future. For example, DIM a$(10,10), z$(10,10): z$=a$.

**-20 READ ONLY (QDOS)-WRITE PROTECTED (SMS)**

This message is normally generated when you are trying to open a channel to a file for the output of data and the medium has been write-protected. Unfortunately, this does not work on microdrives!

**-21 BAD LINE (QDOS)-INVALID SYNTAX (SMS)**

This error message appears if you try to enter a SuperBASIC command which does not make sense. It is in fact a Syntax error - the line will be represented for editing, and if Minerva or SMS is present, the cursor will be (hopefully) placed on top of the offending character.

**-22 IN LINE (QDOS) ***

This message forms part of the error sequence and is used to notify you of the line at which the error occurred (eg. IN LINE 100). This has been modified on Minerva and SMS so that the number of the statement on that line where the error occurred is also shown (eg. IN LINE 100;3).

**-22 UNKNOWN MESSAGE (SMS)**

This error is reported if you try to use REPORT with a number parameter which does not point to an existing message in the computer.

**-23 SECTORS (QDOS) ***

This message is used by commands such as DIR and STAT to signify the number of used sectors/the number of available sectors on the given medium. FORMAT actually uses this to signify the number of available sectors/the number of sectors on the medium.

**-23 ACCESS DENIED (SMS)**

This error message has been implemented for when you try to access files over the network which are protected on the system of the other computer - see SERNET and MIDINET for a further explanation.

**-24, -25 Various messages (QDOS) ***

These messages only appear on the start up screen, to tell you which keys are available to start the QL in different modes. Message -25 is the copyright message explaining who designed the current ROM version. See below.

**-26 DURING WHEN PROCESSING (QDOS) ***

This message is generated after an error message to show that the error has actually occurred within a WHEN definition block.

**-27 PROC/FN CLEARED (QDOS) ***

This message is generated after an error which has occured whilst the interpreter was in the middle of a DEFine PROCedure or DEFine FuNction block. After this message has appeared, any attempt to RETRY or CONTINUE will fail. Also any LOCal variables (or parameters passed to the definition block) will be reset.

Those error messages marked with an asterisk above exist on SMS, but cannot be shown using the REPORT command as they are message groups rather than errors. The equivalents on SMS appear below.

## 43.2 A8.2 Foreign Error Messages

In the following tables, we have tried to list as many of the different language implementations as possible (thanks to QView for this information), although there are bound to be some languages which are not covered here (for example the Russian languages supported on the THOR XVI).

If you can supply any further error messages supported on the QL or its derivatives, then please contact us with a full print-out of the error messages (and preferably, a file on disk with containing the error messages supported).

Under QDOS the error messages are as follows:-

| ER-NUM | English | German | French |
|---|---|---|---|
| -1 | NOT COMPLETE | ABGEBROCHEN | OPERATION NON TERMINÉE |
| -2 | INVALID JOB | FEHLERHAFTER JOB | TACHE INVALIDE |
| -3 | OUT OF MEMORY | SPEICHERUEBERLAUF | HORS CAPACITÉE MEMOIRE |
| -4 | OUT OF RANGE | BEREICHSUEBERLAUF | SORTIE DES LIMITES |
| -5 | BUFFER FULL | PUFFER VOLL | TAMPON PLEIN |
| -6 | CHANNEL NOT OPEN | KANAL NICHT EROEFFNET | CANAL NON OUVERT |
| -7 | NOT FOUND | NICHT GEFUNDEN | NON TROUVÉE |
| -8 | ALREADY EXISTS | EXISTIERT BEREITS | EXISTE DÉJÀ |
| -9 | IN USE | IN BEARBEITUNG | EN USAGE |
| -10 | END OF FILE | DATEIENDE | FIN DE FICHIER |
| -11 | DRIVE FULL | DATENTRAEGER VOLL | LECTEUR PLEIN |
| -12 | BAD NAME | UNGUELTIGE BEZEICHNUNG | NOM INCORRECT |
| -13 | XMIT ERROR | UEBERTRAGUNGS-FEHLER | ERREUR DE TRANSMISSION |
| -14 | FORMAT FAILED | FORMATFEHLER | DEFAILLANCE DANS LE FORMATAGE |
| -15 | BAD PARAMETER | UNGUELTIGER PARAMETER | MAUVAIS PARAMèTRE |
| -16 | BAD OR CHANGED MEDIUM | FEHLERHAFTER DATENTRAEGER | ERREUR DE SUPPORT |
| -17 | ERROR IN EXPRESSION | FEHLER IM AUSDRUCK | ERREUR DANS L'EXPRESSION |
| -18 | OVERFLOW | UEBERLAUF | DÉPASSEMENT DE CAPACITÉ |
| -19 | NOT IMPLEMENTED YET | NICHT IMPLEMENTIERT | COMMANDE NON REPERTORIE … |
| -20 | READ ONLY | NUR LESEN | LECTURE UNIQUEMENT |
| -21 | BAD LINE | SYNTAX-FEHLER | LIGNE INCORRECTE |
| -22 | IN LINE | IN ZEILE | A LA LIGNE NO |
| -23 | SECTORS | SEKTOREN | SECTEURS |
| -26 | DURING WHEN PROCESSING | VERARBEITUNG LÄUFT | PENDANT L'EXECUTION DE WHEN |
| -27 | PROC/FN CLEARED | PROC/FN GELOESCHT | PROC/FN EFFACÉES |

| ERNUM | Swedish | Finnish | Danish |
|---|---|---|---|
| -1 | ej färdig | epätäydellinen | ikke fullf\|rt |
| -2 | fel i jobb | epäkelpo työ | ugyldig Job |
| -3 | minne slut | muisti lopussa | arbeidslager fullt |
| -4 | utom område | ulkopuolella | område overskredet |
| -5 | buffer full | puskuri täynnä | buffer fullt |
| -6 | oöppnad kanal | kanava avaamatta | kanal ikke åpen |
| -7 | hittar ej | ei löydy | ikke funnet |
| -8 | finns redan | jo olemassa | allerede oprettet |
| -9 | används redan | varattu | optatt |
| -10 | fil slut | tiedosto lopussa | filens slutning nådd(EOF) |
| -11 | full kassett | asema täynnä | lagermedie fullt |
| -12 | namnfel | huono nimi | ukjent navn |
| -13 | RS-232 fel | siirtovirhe | transmissjonsfejl |
| -14 | ej formaterbar | alustusvirhe | mislykket formatering |
| -15 | parameterfel | huono parametri | ulovlig parameter |
| -16 | mediafel | huono väline | lese/skrive feil |
| -17 | fel i uttryck | lausekevirhe | feil i utryk |
| -18 | för stort tal | ylitys | numerisk overl\|p |
| -19 | används ej | ei käytössä … | ikke innf\|rt |
| -20 | endast läsning | vain luku kun | lesning tillatt |
| -21 | fel form | huono rivi | feil i linje |
| -22 | På rad | Rivillä | I linje |
| -23 | sektorer | sektoria | sektorer |
| -26 | WHEN under bearbetning | WHEN - rutiinin aikanaved | WHENovervågning |
| -27 | PROC/FN raderad | PROC/FN nollattu | PROC/FN renset |

## 43.3 A8.3 Dates

When defining a new language for use by the computer, not only is it necessary to re-define the error messages, but also the codes used for representing the days of the week and the months of the year.

**Days of the Week**

UK+Finland:Sun Mon Tue Wed Thu Fri Sat

Germany:Son Mon Die Mit Don Fre Sam

France:Dim Lun Mar Mer Jeu Ven Sam

Sweden: Sön Mån Tis Ons Tor Fre Lör

Denmark: Søn Man Tir Ons Tor Fre Lør

**Months of the Year**

UK+Finland:Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

Germany:Jan Feb Mär Apr Mai Jun Jul Aug Sep Okt Nov Dez

France: Jan Fév Mar Avr Mai Jun Jul Aoú Sep Oct Nov Déc

Sweden: Jan Feb Mar Apr Maj Jun Jul Aug Sep Okt Nov Dec

Denmark:Jan Feb Mar Apr Mai Jun Jul Aug Sep Okt Nov Des

## 43.4 A8.4 SMS Messages

Under SMS, the equivalent in-built foreign error messages are as follows:

| ERNUM | German | French |
|---|---|---|
| -1 | unterbrochen | opération incomplète |
| -2 | ungültige Job ID | ID Job non valable |
| -3 | zu wenig freier Speicher | hors capacité mémoire |
| -4 | Wert auáerhalb Bereich | valeur hors limites |
| -5 | puffer voll | tampon plein |
| -6 | ungültige Kanal ID | ID canal non valable |
| -7 | nicht gefunden | est introuvable |
| -8 | existiert bereits | existe déja |
| -9 | wird schon benutztest | utilisé par ailleurs |
| -10 | Datei-Ende | fin de fichier |
| -11 | Medium ist voll | disque plein |
| -12 | ungültiger Name | nom inadmissible |
| -13 | Übertragungs-Fehler | erreur de transmission |
| -14 | Formatier-Fehler | erreur dans le formatage |
| -15 | ungültiger Parameter | paramètre non valable |
| -16 | fehlerhafter Datenträger | erreur de support |
| -17 | Fehler im Ausdruck | erreur dans l'expression |
| -18 | arithmetischer Überlauf | débordement arithmétique |
| -19 | nicht implementiert | ça n'existe pas |
| -20 | schreibgeschützt | protection en écriture |
| -21 | Syntax-Fehler | syntaxe non valable |
| -22 | unbekannte Meldung | message inconnu |
| -23 | Zugriff verweigert | accès interdit |

SMS also incorporates an improved Interpreter, and as a result, has a long list of further errors which can appear either before a program is RUN or whilst a program is RUNning. These errors do not affect ERNUM and do not have an error code as such. Each error is given in English, German and French.

In many instances, these errors replace the QDOS 'Bad Line' error, which left the user to guess why the line had been rejected.

Many of these problems would also be reported if you try to compile the program.

The SBASIC interpreter works in three stages:

**PARSING**

This occurs whenever a new line is entered either from the keyboard as a direct command or using EDIT for example, or when a program is LOADed.

**PRE-COMPILING**

This occurs whenever the command RUN or GO TO is entered - the interpreter runs through the whole of the program to check that structures are correctly defined. It is this stage which has been added to the original QDOS SuperBASIC interpreter and allows SBASIC to be so much quicker than the original.

**RUNNING**

This is the interpreter's job as the program is being RUN - keeping track of variables and program lines, as well as carrying out the actual instructions contained in the program.

Different errors are produced at each stage of the Interpretation process.

**SYNTAX ERROR IN EXPRESSION**

(ENGLISH)

**Syntax-Fehler im Ausdruck**

(German)

**erreur de syntaxe dans l'expression**

(French)

This is reported during PARSING - it normally occurs where you have made a typing error when entering a line and placed two operators together when this is not allowed (or meaningless). For example, the following line will cause this error:

```
x = x ++ 1
```

**MISSING LEFT PARENTHESIS**

(ENGLISH)

**Linke Klammer fehlt**

(German)

**manque parenthèse gauche**

(French)

This error is generated during PARSING - it indicates that there are more closing brackets on a line, than opening brackets. You either need to insert another opening bracket somewhere or delete a closing one.

However, the interpreter reports this error very infrequently - normally 'Invalid Syntax' is reported.

**MISSING RIGHT PARENTHESIS**

(ENGLISH)

**Rechte Klammer fehlt**

(German)

**manque parenthèse droite**

(French)

This message is generated during PARSING - it appears when a program line has more opening brackets than closing brackets.

For example:

```
PRINT CHR$ ((HEX ('d2'))
```

**ERROR IN LINE NUMBER**

(ENGLISH)

**fehlerhafte Zeilennummer**

(German)

**erreur à la ligne numéro**

(French)

This message appears during PARSING - it should be caused whenever you try to enter a line number outside the range 1…32767. However, line numbers which exceed 32767 are merely ignored on current implementations of SMS, causing the program line to be executed as if it had been entered without a line number.

**BAD STRING: MISSING DELIMITER**

(ENGLISH)

**String-Begrenzer fehlt**

(German)

**manque marqueur limite de chaîne**

(French)

This error is reported during PARSING whenever a program line is entered which contains a string within quote marks (either single or double) and one of those quote marks is missing.

Example:

```
PRINT 'Hello "There"
```

**INCORRECT PROCEDURE OR FUNCTION DEFINITION**

(ENGLISH)

**falsche Definition einer Prozedur oder Funktion**

(German)

**mauvaise définition d'une procédure ou fonction**

(French)

This message is reported during PARSING and indicates that there is something amiss with a program line containing DEFine PROCedure or DEFine FuNction, for example where one of the end brackets is missing around the parameter definition list, or one of the parameters appears in the definition as just a comma or empty quotes:

```
1 DEFine PROCedure TEST (a,"")
```

and:

```
1 DEFine PROCedure TEST (a,)
```

both produce this error.

Other problems may be indicated by the error 'Invalid Syntax', such as no opening bracket appearing before the list of parameters.

**PROCEDURE OR FUNCTION DEFINITION NOT ALLOWED HERE**

(ENGLISH)

**Prozedur- oder Funktion-Definition hier nicht erlaubt**

(German)

**définition d'une fonction ou procédure non permise ici**

(French)

This message is reported during PARSING and occurs if you try to enter a line containing the DEFine PROCedure or DEFine FuNction structure as a direct command (rather than as a program line).

**DEFINES MAY NOT BE WITHIN OTHER CLAUSES**

(ENGLISH)

**DEFines dürfen nicht innerhalb Strukturen stehen**

(German)

**DEFines ne peuvent se trouver dans d'autres structures**

(French)

This message is reported during PRE-COMPILING if the program includes a line containing DEFine PROCedure or DEFine FuNction inside another structure, such as another DEFine … END DEFine clause, or SELect … END SELect structure, IF … END IF, WHEN … END WHEN.

Unfortunately, a lot of very early SuperBASIC programs written for the Sinclair QL fall foul of this rule. The old style interpreter would jump the rogue DEFine structure, sometimes falling out of the program because the problem was actually a missing END DEFine statement.

**MISPLACED END DEFINE**

(ENGLISH)

**END DEFine darf hier nicht stehen**

(German)

**END DEFine n'est pas à sa place ici**

(French)

This error is reported during PRE-COMPILING if a program line contains END DEFine without a relative DEFine PROCedure or DEFine FuNction.

**MISPLACED LOCAL**

(ENGLISH)

**LOCal darf hier nicht stehen**

(German)

**LOCal n'est pas à sa place ici**

(French)

This message is reported during PRE-COMPILING if the program contains a LOCal statement other than as the first active program line after a DEFine PROCedure or DEFine FuNction statement.

**RETURN NOT IN PROCEDURE OR FUNCTION**

(ENGLISH)

**RETurn ist nicht innerhalb Prozedur oder Funktion**

(German)

**RETurn ne se trouve pas dans une fonction ou procédure**

(French)

This message is generated during RUNNING if the interpreter tries to execute a RETurn command outside of a DEFine PROCedure or DEFine FuNction structure.

It will also be reported during RUNNING if the interpreter is executing a DEFine FuNction structure, but meets an END DEFine statement - in other words, the RETurn command is missing from the structure.

**WHEN CLAUSES MAY NOT BE NESTED**

(ENGLISH)

**WHEN Strukturen dürfen nicht verschachtelt sein**

(German)

**des structures WHEN ne peuvent être emboîtées**

(French)

This error is generated during PRE-COMPILING if a program contains a WHEN ERRor (or WHEN variable, if implemented) structure inside another one.

**MISPLACED END WHEN**

(ENGLISH)

**END WHEN darf hier nicht stehen**

(German)

**END WHEN n'est pas à sa place ici**

(French)

This error is generated during PRE-COMPILING if the program contains an END WHEN statement without a corresponding WHEN ERRor or WHEN variable statement.

**MISPLACED ELSE**

(ENGLISH)

**ELSE darf hier nicht stehen**

(German)

**ELSE n'est pas à sa place ici**

(French)

This error is generated during PRE-COMPILING if the program contains an ELSE statement without a corresponding IF statement.

**MISPLACED END IF**

(ENGLISH)

**END IF darf hier nicht stehen**

(German)

**END IF n'est pas à sa place ici**

(French)

This error is generated during PRE-COMPILING if the program contains an END IF statement without a corresponding IF statement.

**PROGRAM STRUCTURES NESTED TOO DEEPLY, MY BRAIN ACHES**

(ENGLISH)

**Strukturen zu tief verschachtelt**

(German)

**les structures sont trop emboîtées, ça me fait mal au crÀne**

(French)

This message will rarely appear - it will be generated during RUNNING if the program uses PROCedures or FuNctions which call themselves too many times.

You are in fact more likely to run out of memory or crash the machine than see this message!!

**INCOMPLETE IF CLAUSE**

(ENGLISH)

**unvollständige IF Struktur**

(German)

**structure IF incomplète**

(French)

This error is generated during PRE-COMPILING if the program contains an IF statement without a corresponding END IF statement.

NOTE that in-line IF structures do not necessarily need a corresponding END IF statement.

**INCOMPLETE SELECT CLAUSE**

(ENGLISH)

**unvollständige SELect Struktur**

(German)

**structure SELECT incomplète**

(French)

This error is generated during PRE-COMPILING if the program contains a SELect ON statement without a corresponding END SELect statement.

NOTE that in-line SELect ON structures do not necessarily need a corresponding END SELect statement.

**INCOMPLETE DEFINE**

(ENGLISH)

**unvollständiges DEFine**

(German)

**structure DEFINE incomplète**

(French)

This error is generated during PRE-COMPILING if the program contains a DEFine PROCedure statement or a DEFine FuNction statement without a corresponding END DEFine statement.

**INCOMPLETE WHEN CLAUSE**

(ENGLISH)

**unvollständige WHEN Struktur**

(German)

**structure WHEN incomplète**

(French)

This error is generated during PRE-COMPILING if the program contains a WHEN ERRor statement (or WHEN variable when supported) without a corresponding END WHEN statement.

**UNACCEPTABLE LOOP VARIABLE**

(ENGLISH)

**unerlaubte Schleifen-Variable**

(German)

**variable de contrôle boucle inacceptable**

(French)

This message appears during the PARSING stage if a program line contains a FOR loop with a string loop identifier (compare Minerva), such as:

```
FOR a$='a' TO 'z'
```

**UNABLE TO FIND AN OPEN LOOP**

(ENGLISH)

**kann keine offene Schleife finden**

(German)

**aucune boucle ouverte ne peut être trouvée**

(French)

This message appears during the PRE-COMPILING phase if a program contains an EXIT, NEXT, END FOR or END REPeat statement which does not have a loop control variable specified (compare 'Undefined Loop Control Variable') and the Interpreter is unable to find a corresponding FOR or REPeat statement.

**UNDEFINED LOOP CONTROL VARIABLE**

(ENGLISH)

**undefinierte Schleifen-Variable**

(German)

**la variable de contrôle boucle est indéfinie**

(French)

This message is similar to 'Unable to Find an Open Loop' except that it appears during RUNNING if a program contains an EXIT, NEXT, END FOR or END REPeat statement which includes the name of a loop control variable and the Interpreter is unable to find a corresponding FOR or REPeat statement.

This will also happen if the loop control variable has been re-defined before the EXIT, NEXT, END FOR or END REPeat statement is executed, for example:

```
FOR x=1 to 100
...
...
DIM x(100)
...
...
END FOR x
```

**MISPLACED END SELECT**

(ENGLISH)

**END SELect darf hier nicht stehen**

(German) (French)

**END SELect n'est pas à sa place ici**

(French)

This message appears during PRE-COMPILING if a program contains an END SELect statement without a corresponding SELect ON statement.

**DATA IN COMMAND LINE HAS NO MEANING**

(ENGLISH)

**DATA in Befehlszeige wird ignoriert**

(German)

**DATA dans une ligne de commande n'a pas de sens**

(French)

This message appears during PARSING if a line containing a DATA statement is entered as a direct command.

**INCORRECTLY STRUCTURED SELECT CLAUSE**

(ENGLISH)

**falsch strukturiertes SELect**

(German)

**SELECT mal structuré**

(French)

This message is generated during PRE-COMPILING in one of two cases:

- a SELect ON statement appears without any comparison values, such as:

```
10 SELect ON x
20 PRINT 'Hello'
30 END SELect
```

- the comparison values appear in a program outside of a SELect ON structure, for example:

```
10 SELect ON x
20 =10 : PRINT 'x=10'
30 END SELect
40 =20 : PRINT 'x=20'
```

**UNACCEPTABLE PARAMETERS FOR READ**

(ENGLISH)

**unerlaubte Parameter für READ**

(German)

**paramètre inacceptable pour READ**

(German) (French)

This message appears during PRE-COMPILING if a READ statement has meaningless parameters, for example:

```
READ 'x'
READ s,s1,s*1
READ 1,1,2
```

Note however, that no error is caused by the READ statement without any parameters.

Compare also:

READ PRINT

which causes an error during RUNNING - assignment can only be a variable or array element.

**END OF DATA**

(ENGLISH)

**Ende von DATA**

(German)

**fin de DATA**

(French)

This message is generated during RUNNING if a program is trying to READ DATA statements but has run out of DATA to read - use RESTORE or add check that all of the required DATA is contained in the program.

**SBASIC CANNOT PERFORM READS WITHIN DATA EXPRESSIONS**

(ENGLISH)

**SBASIC kann keine READs innerhalb DATAs ausführen**

(German)

**SBASIC ne peut effectuer des READs dans des expressions DATA**

(French)

We are uncertain when this error appears, not having been able to create a situation which causes this error to be reported.

If a DATA statement contains a procedure name, such as:

```
DATA 1,1,READ
```

or:

```
DATA 1,1,PRINT
```

then during RUNNING, when the program tries to read the data parameter 'READ', the error 'unknown function or array' is produced.

**UNKNOWN PROCEDURE**

(ENGLISH)

**unbekannte Prozedur**

(German)

**procédure inconnue**

(French)

This message is displayed during RUNNING if a procedure name is used which has not been defined - this normally suggests one of three problems:

- a typing error
- a machine code toolkit has not been linked in properly
- a SuperBasic DEFine PROCedure structure is missing.

**UNKNOWN FUNCTION OR ARRAY**

(ENGLISH)

**unbekannte Funktion oder Feld**

(German)

**fonction ou tableau inconnus**

(French)

This message is displayed during RUNNING if a Procedure name has been used as a function, variable or array descriptor. This normally suggests that a program uses the same name for a variable as a toolkit which has been linked in.

**ONLY ARRAYS MAY BE DIMENSIONED**

(ENGLISH)

**nur Felder dürfen dimensioniert werden**

(German)

**on ne peut dimensionner que des tableaux**

(French)

This message is displayed during RUNNING if a Procedure name has been used as an array name in a DIM statement. This normally suggests that a program uses the same name for a variable as a toolkit which has been linked in.

This error is also reported if you try to DIMension the name of a parameter passed to a PROCedure or FuNction, such as:

```
100 DEFine PROCedure TEST(x)
110 DIM x(100)
120 END DEFine
```

- Use LOCal instead, such as:

```
110 LOCal x(100)
```

(although why you would want to do this, is anyone's guess!!)

**PROCEDURE AND FUNCTION PARAMETERS MAY NOT BE DIMENSIONED**

(ENGLISH)

**Prozedur- oder Funktion-Parameter dürfen nicht dimensioniert werden**

(German)

**les paramètres des procédures et fonctions ne peuvent être dimensionnés**

(French)

This error is intended to trap the second example for 'Only Arrays May be Dimensioned' - see description of DIM for an example and the difference between these two errors.

**SBASIC CANNOT PUT UP WITH NEGATIVE DIMENSIONS**

(ENGLISH)

**SBASIC mag keine negativen Dimensionen**

(German)

**SBASIC ne sait comment traiter des dimensions négatives**

(French)

This error is reported during RUNNING if you try to DIMension an array with a negative index, such as:

DIM x(-100)

Note that if you try to use a negative index in other situations, such as:

x(-100)=32

the error 'Array Index out of Range' will be reported.

**DIMENSIONAL OVERFLOW - YOU CANNOT BE SERIOUS!**

(ENGLISH)

**Dimensions-Überlauf**

(German)

**dépassement de dimension - soyons sérieux!**

(French)

This message appears during RUNNING if you try to DIMension an array with too many indices - this appears to happen after around 7 indices). For example, the error will be caused by the following line:

DIM x(1,2,3,4,5,6,7,8)

NOTE the warning listed below!!

**ERROR IN INDEX LIST**

(ENGLISH)

**Fehler in Index-Liste**

(German)

**erreur dans la liste d'indexage**

(French)

We are uncertain when this error appears, not having been able to create a situation which causes this error to be reported.

**TOO MANY INDEXES**

(ENGLISH)

**zu viele Indizes**

(German)

**trop d'indices**

(French)

We are uncertain when this error appears, not having been able to create a situation which causes this error to be reported.

**CANNOT ASSIGN TO SUB-ARRAY**

(ENGLISH)

**kann nicht auf Teil-Feld zuweisen**

(German)

**impossible d'assigner à un sous-tableau**

(French)

We are uncertain when this error appears, not having been able to create a situation which causes this error to be reported.

**UNACCEPTABLE ARRAY INDEX LIST**

(ENGLISH)

**fehlerhafte Feld-Index-Liste**

(German)

**liste d'indices dans tableau inacceptab (French)le**

(French)

This message is generated during RUNNING if you try to use an array with more indices that it was DIMensioned with, for example:

```
DIM x(100,100)
x(10,10,10)=52
```

The error can also be generated when you try to assign a value across several array elements at a time (this should possibly cause the error 'Cannot Assign to a Sub-Array'), for example:

```
DIM x(10,10)
x(3,4 TO 5)=100
```

Beware of the dangers here - see below

**WARNINGS:**

In current versions of SMSQ/E, if you DIMension an array with the maximum number of indices and use two more indices in the reference that this maximum, you can crash the computer, for example:

```
DIM x(1,2,3,4,5,6,7)
x(1,2,3,4,5,6,7,8,9)=52
```

You can also crash the computer if you miss out array indexes:

x(1,,1)=100

Another way of crashing the computer is when trying to assign a value across several array elements, for example:

x(1 TO 3,10)=52

**ARRAY INDEX OUT OF RANGE**

(ENGLISH)

**Feld-Index auáerhalb Bereich**

(German)

**indice tableau hors limites**

(French)

This message is generated during RUNNING if the value of an index specified in an array is higher than that specified when the array was DIMensioned, for example:

```
DIM x(10,10)
x(10,12)=52
```

Note however, that if you try to use an index which exceeds 32767, the error 'Error in Expression' is generated.

**ONLY ARRAYS OR STRINGS MAY BE INDEXED**

(ENGLISH)

**nur Felder oder Strings dürfen indiziert werden**

(German)

**on peut indexer uniquement des tableaux ou chaînes**

(French)

This message is generated during RUNNING if you try to reference an array which has not yet been DIMensioned, for example:

CLEAR

x(100)=52

Compare the situation where you try to index a name which is in fact defined as a Procedure - the index is ignored and the Procedure executed as normal, for example

PRINT (100)=32 displays 32 on the screen.

On the other hand, if you try to index a name which is defined as a Function, the error 'Unknown Procedure' is generated instead.

In both cases, compare what happens when an index is not specified (the next error listed here is generated).

**ASSIGNMENT CAN ONLY BE TO A VARIABLE OR ARRAY ELEMENT**

(ENGLISH)

**Zuweisungen nur an Variable oder Feld-Element**

(German)

**assignation uniquement vers une variable ou un élément d'un tableau**

(French)

This error is generated during RUNNING when a program tries to assign a value to a variable which is actually defined as a Procedure or Function already (this suggests that a toolkit may have re-defined a variable name).

**MISTAKE IN PROGRAM**

(ENGLISH)

**MISTake - Fehler im Programm**

(German)

**MISTake - Erreur de programmation**

(French)

This message is generated during PRE-COMPILING - whilst a program is being LOADed (or QLOADed), if a line has generated an error during PARSING, the word MISTake is inserted in the relevant line in the program. This message is generated if you try to RUN the program without altering the offending line.

**DURING WHEN PROCESSING**

(ENGLISH)

**während WHEN-Bearbeitung**

(German)

**pendant le traitement de when**

(French)

This message is generated during RUNNING if an error occurs whilst the program was executing a WHEN ERRor (or WHEN variable when it is implemented) structure. You should enter WHEN ERRor as a direct command to switch off the WHEN ERRor trapping.

**PROC/FN CLEARED**

(ENGLISH)

**PROC/FN gelöscht**

(German)

**PROC/FN effacée**

(French)

If an error is generated whilst the program is executing a DEFine PROCedure or DEFine FuNction structure, this error will be generated when you EDIT the program, or enter CONTINUE. Unlike earlier ROMs, this does not seem to prevent you from using CONTINUE to carry on with RUNning the program from the place the error occurred.

**At line**

(ENGLISH)

**In Zeile**

(German)

**A la ligne**

(French)

This is merely the message used to generate part of all error messages, signifying the line number and statement number where the error occurred.

**FATAL ERROR IN SBASIC INTERPRETER**

(ENGLISH)

**schwerwiegender Fehler im SBASIC-Interpreter**

(German)

**erreur fatale dans l'interpréteur SBASIC**

(French)

This message should hopefully never happen - it means that the interpreter has become corrupt. If a multiple SBASIC interpreter, it will be removed from the system when you press a key.

One instance where this error will occur is if you try to RUN a program which has been QLOADed and the original file was created using QSAVE on a Minerva ROM with integer tokenisation enabled.

CHAPTER

FORTYFOUR

# A9 CHARACTER SET, KEYBOARD

This Appendix deals with the QL character set and keyboard.

## 44.1 A9.1 The Character Set

The codes from 0 to 31 (except 10) and 192 to 254 are not standardised - they are undefined on standard QLs (a checkerboard square) but Minerva sets them to the characters listed below. The keying required for many of the characters seems to differ on every type of machine, which is very annoying for the programmer.

Some of the keyings will also be different on new replacement keyboards, although these normally only add to the standard set.

We have supplied the keyings on standard British and German QLs so that you can see where problems may occur in different keyings. We recommend that you make your program fully configurable if these differences appear to be a problem.

The full character set is:

| Code | Character | Keys to press |
|---|---|---|
| 0 | NUL | <CTRL> <£> British <CTRL> < ESC> German |
| 1 | F1 | <CTRL> <A> |
| 2 | F2 | <CTRL> <B> |
| 3 | F3 | <CTRL> <C> |
| 4 | F4 | <CTRL> <D> |
| 5 | F5 | <CTRL> <E> |
| 6 | AK | <CTRL> <F> |
| 7 | minim | <CTRL> <G> |
| 8 | BS | <CTRL> <H> |
| 9 | HT | <CTRL> <I> <TAB> |
| 10 | NL | <CTRL> <J> <ENTER> |
| 11 | VT | <CTRL><K> |
| 12 | FF | <CTRL><L> |
| 13 | CR | <CTRL><M> |
| 14 | SO | <CTRL><N> |
| 15 | SI | <CTRL><O> |
| 16 | $^{0}$ (small 0) | <CTRL><P> |
| 17 | $^{1}$ (small 1) | <CTRL><Q> |
| 18 | $^{2}$ (small 2) | <CTRL><R> |

continues on next page

Table 1 – continued from previous page

| Code | Character | Keys to press |
|---|---|---|
| 19 | 3 (small 3) | <CTRL><S> |
| 20 | 4 (small 4) | <CTRL><T> |
| 21 | 5 (small 5) | <CTRL><U> |
| 22 | 6 (small 6) | <CTRL><V> |
| 23 | 7 (small 7) | <CTRL><W> |
| 24 | 8 (small 8) | <CTRL><X> |
| 25 | 9 (small 9) | <CTRL><Y> |
| 26 | A (small A) | <CTRL><Z> |
| 27 | B (small B) | <ESC>, <CTRL><SHIFT><[> British <CTRL><SHIFT><Ü> German |
| 28 | C (small C) | <CTRL><SHIFT><\> British <CTRL><SHIFT><<> German |
| 29 | D (small D) | <CTRL><SHIFT><]> British <CTRL><SHIFT><+> German |
| 30 | E (small E) | <CTRL><SHIFT><£> British <CTRL><SHIFT><\> German |
| 31 | F (small F) | <CTRL><SHIFT><ESC> |
| 32 | (space) | <SPACE> |
| 33 | ! (exclamation) | <SHIFT><1> |
| 34 | “ (speech mark) | <SHIFT><’> British <SHIFT><2> German |
| 35 | # (hash) | <SHIFT><3> British <#> German |
| 36 | $ (dollar) | <SHIFT><4> |
| 37 | % (percent) | <SHIFT><5> |
| 38 | & (ampersand) | <SHIFT><7> British <SHIFT><6> German |
| 39 | ‘ (quote) | <’> British <SHIFT><#> German |
| 40 | ( (bracket) | <SHIFT><9> British <SHIFT><8> German |
| 41 | ) (bracket) | <SHIFT><0> British <SHIFT><9> German |
| 42 | * (asterisk) | <SHIFT><8> British <SHIFT><+> German |
| 43 | • (plus) | <SHIFT><=> British <+> German |
| 44 | , (comma) | <,> |
| 45 | • (minus/hyphen) | <-> |
| 46 | . (fullstop) | <.> |
| 47 | / (stroke) | </> British <SHIFT><7> German |
| 48 | 0 | <0> |
| 49 | 1 | <1> |
| 50 | 2 | <2> |
| 51 | 3 | <3> |
| 52 | 4 | <4> |

continues on next page

Table 1 – continued from previous page

| Code | Character | Keys to press |
|---|---|---|
| 53 | 5 | <5> |
| 54 | 6 | <6> |
| 55 | 7 | <7> |
| 56 | 8 | <8> |
| 57 | 9 | <9> |
| 58 | : (colon) | <SHIFT><;> British<br><SHIFT><.> German |
| 59 | ; (semicolon) | <;> British <SHIFT><,> German |
| 60 | < (less than) | <SHIFT><,> British < < > German |
| 61 | = (equal) | <=> British <SHIFT><=> German |
| 62 | > (greater) | <SHIFT><.> British <SHIFT><< > German |
| 63 | ? (question mark) | <SHIFT></> British<br><SHIFT><á> German |
| 64 | @ (address symbol) | <SHIFT><2> British<br><CTRL><\> German |
| 65 | A | <SHIFT><A> |
| 66 | B | <SHIFT><B> |
| 67 | C | <SHIFT><C> |
| 68 | D | <SHIFT><D> |
| 69 | E | <SHIFT><E> |
| 70 | F | <SHIFT><F> |
| 71 | G | <SHIFT><G> |
| 72 | H | <SHIFT><H> |
| 73 | I | <SHIFT><I> |
| 74 | J | <SHIFT><J> |
| 75 | K | <SHIFT><K> |
| 76 | L | <SHIFT><L> |
| 77 | M | <SHIFT><M> |
| 78 | N | <SHIFT><N> |
| 79 | O | <SHIFT><O> |
| 80 | P | <SHIFT><P> |
| 81 | Q | <SHIFT><Q> |
| 82 | R | <SHIFT><R> |
| 83 | S | <SHIFT><S> |
| 84 | T | <SHIFT><T> |
| 85 | U | <SHIFT><U> |
| 86 | V | <SHIFT><V> |
| 87 | W | <SHIFT><W> |
| 88 | X | <SHIFT><X> |
| 89 | Y | <SHIFT><Y> |
| 90 | Z | <SHIFT><Z> |
| 91 | [ (square bracket) | <[> British <CTRL><9> German |
| 92 | \ (backslash) | <\> |
| 93 | ] (square bracket) | <]> British <CTRL><0> German |
| 94 | ^ (circumflex) | <SHIFT><6> British<br><SHIFT><\> German |
| 95 | _ (underscore) | <SHIFT><-> |

continues on next page

Table 1 – continued from previous page

| Code | Character | Keys to press |
|---|---|---|
| 96 | £ (pound) | <£> British <CTRL><7> German |
| 97 | a | <A> |
| 98 | b | <B> |
| 99 | c | <C> |
| 100 | d | <D> |
| 101 | e | <E> |
| 102 | f | <F> |
| 103 | g | <G> |
| 104 | h | <H> |
| 105 | i | <I> |
| 106 | j | <J> |
| 107 | k | <K> |
| 108 | l | <L> |
| 109 | m | <M> |
| 110 | n | <N> |
| 111 | o | <O> |
| 112 | p | <P> |
| 113 | q | <Q> |
| 114 | r | <R> |
| 115 | s | <S> |
| 116 | t | <T> |
| 117 | u | <U> |
| 118 | v | <V> |
| 119 | w | <W> |
| 120 | x | <X> |
| 121 | y | <Y> |
| 122 | z | <Z> |
| 123 | { (brace) | <SHIFT><[> British<br><CTRL><á> German |
| 124 | \| (vertical line) | <SHIFT><\> British<br><CTRL><8> German |
| 125 | } (brace) | <SHIFT><]> British<br><CTRL><#> German |
| 126 | ~ (tilde) | <SHIFT><£> British<br><CTRL><#> German |
| 127 | copyright | <SHIFT><ESC> |
| 128 | a umlaut | <CTRL><ESC> British <Ä> German |
| 129 | a tilde | <CTRL><SHIFT><1> |
| 130 | a circle | <CTRL><SHIFT><’> British<br><CTRL><SHIFT><Ä> German |
| 131 | e acute | <CTRL><SHIFT><3> |
| 132 | o umlaut | <CTRL><SHIFT><4> British<br><Ö> German |
| 133 | o tilde | <CTRL><SHIFT><5> |
| 134 | o bar | <CTRL><SHIFT><7> |
| 135 | u umlaut | <CTRL><’> British <Ü> German |
| 136 | c cedilla | <CTRL><SHIFT><9> |

continues on next page

Table 1 – continued from previous page

| Code | Character | Keys to press |
|---|---|---|
| 137 | n tilde | <CTRL><SHIFT><0> |
| 138 | ae diphthong | <CTRL><SHIFT><8> |
| 139 | oe diphthong | <CTRL><SHIFT><=> British<br><CTRL><SHIFT><#> German |
| 140 | a acute | <CTRL><,> |
| 141 | a grave | <CTRL><-> British<br><CTRL><SHIFT><4> German |
| 142 | a circumflex | <CTRL><.> |
| 143 | e umlaut | <CTRL></> British <CTRL><-> German |
| 144 | e grave | <CTRL><0> British<br><CTRL><SHIFT><V> German |
| 145 | e circumflex | <CTRL><1> |
| 146 | i umlaut | <CTRL><2> |
| 147 | i acute | <CTRL><3> |
| 148 | i grave | <CTRL><4> |
| 149 | i circumflex | <CTRL><5> |
| 150 | o acute | <CTRL><6> |
| 151 | o grave | <CTRL><7> British<br><CTRL><SHIFT><,> German |
| 152 | o circumflex | <CTRL><8> British<br><CTRL><SHIFT><D> German |
| 153 | u acute | <CTRL><9> British<br><CTRL><Ä> German |
| 154 | u grave | <CTRL><SHIFT><;> British<br><CTRL><SHIFT><Ö> German |
| 155 | u circumflex | <CTRL><;> British<br><CTRL><Ö> German |
| 156 | á (beta/sz) | <CTRL><SHIFT><,> British<br><á> German |
| 157 | cent symbol | <CTRL><=> British<br><CTRL><SHIFT><G> German |
| 158 | yen symbol | <CTRL><SHIFT><.> |
| 159 | backquote | <CTRL><SHIFT></> British<br><CTRL><SHIFT><-> German |
| 160 | A umlaut | <CTRL><SHIFT><2> British<br><SHIFT><Ä> German |
| 161 | A tilde | <CTRL><SHIFT><A> |
| 162 | A circle | <CTRL><SHIFT><B> |
| 163 | E acute | <CTRL><SHIFT><C> |
| 164 | O umlaut | <CTRL><SHIFT><D> British<br><SHIFT><Ö> German |
| 165 | O tilde | <CTRL><SHIFT><E> |
| 166 | O bar | <CTRL><SHIFT><F> |
| 167 | U umlaut | <CTRL><SHIFT><G> British<br><SHIFT><Ü> German |
| 168 | C cedilla | <CTRL><SHIFT><H> |
| 169 | N tilde | <CTRL><SHIFT><I> |

continues on next page

Table 1 – continued from previous page

| Code | Character | Keys to press |
|---|---|---|
| 170 | AE diphthong | <CTRL><SHIFT><J> |
| 171 | OE diphthong | <CTRL><SHIFT><K> |
| 172 | lower alpha | <CTRL><SHIFT><L> |
| 173 | lower delta | <CTRL><SHIFT><M> |
| 174 | upper theta | <CTRL><SHIFT><N> |
| 175 | lower lambda | <CTRL><SHIFT><O> |
| 176 | lower mu | <CTRL><SHIFT><P> |
| 177 | lower pi | <CTRL><SHIFT><Q> |
| 178 | upper phi | <CTRL><SHIFT><R> |
| 179 | inverse ! | <CTRL><SHIFT><S> |
| 180 | inverse ? | <CTRL><SHIFT><T> |
| 181 | script mark | <CTRL><SHIFT><U> |
| 182 | section symbol | <CTRL><SHIFT>(V):sup:*British*<br><SHIFT><3> German |
| 183 | cross-circle | <CTRL><SHIFT><W> |
| 184 | French quote | <CTRL><SHIFT><X> |
| 185 | French quote | <CTRL><SHIFT><Y> British<br><CTRL><SHIFT><Z> German |
| 186 | ø (degree) | <CTRL><SHIFT><Z> British<br><CTRL><SHIFT><Y> German |
| 187 | division | <CTRL><[> British<br><CTRL><Ü> German |
| 188 | left arrow | <CTRL><\> British<br><CTRL><SHIFT><2> German |
| 189 | right arrow | <CTRL><]> British<br><CTRL><+> German |
| 190 | up arrow | <CTRL><SHIFT><6> |
| 191 | down arrow | <CTRL><SHIFT><-> British<br><CTRL><SHIFT><á> German |
| 192 | up-left arrow | <LEFT> |
| 193 | up-right arrow | <ALT><LEFT> |
| 194 | down-left arrow | <CTRL><LEFT> |
| 195 | down-right arrow | <CTRL><ALT><LEFT> |
| 196 | upper delta | <SHIFT><LEFT> |
| 197 | small eta | <ALT><SHIFT><LEFT> |
| 198 | large upper phi | <CTRL><SHIFT><LEFT> |
| 199 | upper gamma | <ALT><CTRL><SHIFT><LEFT> |
| 200 | spades | <RIGHT> |
| 201 | hearts | <ALT><RIGHT> |
| 202 | diamonds | <CTRL><RIGHT> |
| 203 | clubs | <ALT><CTRL><RIGHT> |
| 204 | upper lambda | <SHIFT><RIGHT> |
| 205 | inverse upper delta | <ALT><SHIFT><RIGHT> |
| 206 | infinity | <CTRL><SHIFT><RIGHT> |
| 207 | upper omega | <ALT><CTRL><SHIFT><RIGHT> |
| 208 | upper pi | <UP> |
| 209 | upper psi | <ALT><UP> |
| 210 | registered | <CTRL><UP> |

continues on next page

Table 1 – continued from previous page

| Code | Character | Keys to press |
|---|---|---|
| 211 | upper sigma | <ALT><CTRL><UP> |
| 212 | upper theta | <SHIFT><UP> |
| 213 | upper upsilon | <ALT><SHIFT><UP> |
| 214 | dagger | <CTRL><SHIFT><UP> |
| 215 | double dagger | <ALT><CTRL><SHIFT><UP> |
| 216 | upper xi | <DOWN> |
| 217 | plus minus | <ALT><DOWN> |
| 218 | [unknown] | <CTRL><DOWN> |
| 219 | exactly equal | <ALT><CTRL><DOWN> |
| 220 | less or equal | <SHIFT><DOWN> |
| 221 | not equal | <ALT><SHIFT><DOWN> |
| 222 | greater or equal | <CTRL><SHIFT><DOWN> |
| 223 | approximately equal | <ALT><CTRL><SHIFT><DOWN> |
| 224 | empty square | <CAPS> |
| 225 | filled square | <ALT><CAPS> |
| 226 | filled circle | <CTRL><CAPS> |
| 227 | lower chi | <ALT><CTRL><CAPS> |
| 228 | differential/del | <SHIFT><CAPS> |
| 229 | element-of | <ALT><SHIFT><CAPS> |
| 230 | FR (French Francs) | <CTRL><SHIFT><CAPS> |
| 231 | lower gamma | <ALT><CTRL><SHIFT><CAPS> |
| 232 | upper kappa | <F1> |
| 233 | lower iota | <CTRL><F1> |
| 234 | vertical line | <SHIFT><F1> |
| 235 | lower kappa | <CTRL><SHIFT><F1> |
| 236 | one-quarter | <F2> |
| 237 | one-half | <CTRL><F2> |
| 238 | three-quarters | <SHIFT><F2> |
| 239 | lower omega | <CTRL><SHIFT><F2> |
| 240 | lower psi | <F3> |
| 241 | => (conclusion) | <CTRL><F3> |
| 242 | lower rho | <SHIFT><F3> |
| 243 | lower sigma | <CTRL><SHIFT><F3> |
| 244 | lower tau | <F4> |
| 245 | lower upsilon | <CTRL><F4> |
| 246 | square root | <SHIFT><F4> |
| 247 | cubic root | <CTRL><SHIFT><F4> |
| 248 | lower xi | <F5> |
| 249 | … (three dots) | <CTRL><F5> |
| 250 | lower zeta | <SHIFT><F5> |
| 251 | integral head | <CTRL><SHIFT><F5> |
| 252 | integral middle | <SHIFT><SPACE> |
| 253 | integral foot | <SHIFT><TAB> |
| 254 | random pattern | <SHIFT><ENTER> |
| 255 | regular pattern | ( <ALT> ) |

*) Yes, there are in fact three kinds of phi!

Note that CHR$(255) is used whenever a character code is undefined.

To make matters worse, the JS ROM makes a distinction between <CTRL><ESC> with and without caps lock (the former is equivalent to <CTRL><SHIFT><ESC>.

To see Minerva's pattern character assigned to CHR$(255), try the following lines. Changing the CHAR_INC parameters is fun, or add OVER 1 at line 100…

```
100 CHAR_INC 6,9: CLS
110 FOR i=1 TO 10: PRINT FILL$(CHR$(255),30)
```

If you are designing a program which should work (without annoying the user) on all QLs and keyboards then avoid all of the above codes which are shown as being different on British and German QLs. Alternatively, you could make your program fully configurable for these keys, or store all of the different keyboard layouts in your program so that it will adapt itself to the machine on which it is working - the latter is however much work.

## 44.2 A9.2 Keyboard Layouts

As more and more QL Emulators appear as well as keyboard interfaces, it becomes increasingly difficult to list the standard keyboard layouts which will be available to a QL user. Instead, we have listed here the layouts used in the most common countries on a standard QL machine (as supplied by Sinclair).

### 44.2.1 British QL

```
F1     ESC  1   2   3   4   5   6   7   8   9   0   -   =   £   \
F2     TAB    Q   W   E   R   T   Y   U   I   O   P   [   ]
F3     CAPS    A   S   D   F   G   H   J   K   L   ;   '       ENTER
F4     SHIFT     Z   X   C   V   B   N   M   ,   .   /     SHIFT
F5     CTRL  LEFT RIGHT          SPACE             UP  DOWN   ALT
```

### 44.2.2 German QL

```
F1     ESC  1   2   3   4   5   6   7   8   9   0   á   #   \   <
F2     TAB    Q   W   E   R   T   Z   U   I   O   P   Ü   +
F3     CAPS    A   S   D   F   G   H   J   K   L   Ö   Ä       ENTER
F4     SHIFT     Y   X   C   V   B   N   M   ,   .   -     SHIFT
F5     CTRL  LEFT RIGHT          SPACE             UP  DOWN   ALT
```

### 44.2.3 French QL

```
F1     ESC  1   2   3   4   5   6   7   8   9   0   -   =   ^   ù
F2     TAB    A   Z   E   R   T   Y   U   I   O   P   é   è
F3     CAPS    Q   S   D   F   G   H   J   K   L   M   à       ENTER
F4     SHIFT     W   X   C   V   B   N   ,   .   ;   ç     SHIFT
F5     CTRL  LEFT RIGHT          SPACE             UP  DOWN   ALT
```

### 44.2.4 Swedish QL

```
F1     ESC  1   2   3   4   5   6   7   8   9   0   +   '   |   <
F2     TAB    Q   W   E   R   T   Y   U   I   O   P   Å   *
F3     CAPS    A   S   D   F   G   H   J   K   L   Ö   Ä       ENTER
```

(continues on next page)

(continued from previous page)

```
F4      SHIFT     Z   X   C   V   B   N   M   ,   .   -     SHIFT
F5      CTRL  LEFT RIGHT          SPACE             UP  DOWN   ALT
```

### 44.2.5 Finnish QL

```
F1      ESC  1   2   3   4   5   6   7   8   9   0   +   '   |   <
F2      TAB    Q   W   E   R   T   Y   U   I   O   P   Å   ^
F3      LUKITUS A   S   D   F   G   H   J   K   L   Ö   Ä      ENTER
F4      VAIHTO   Z   X   C   V   B   N   M   ,   .   -      VAIHTO
F5      OHJAUS LEFT RIGHT         SPACE            UP DOWN     SIIRTO
```

### 44.2.6 Danish QL

```
F1      ESC  1   2   3   4   5   6   7   8   9   0   =   +   /  DEL
F2      TAB    Q   W   E   R   T   Y   U   I   O   P   Å   "
F3      LOCK    A   S   D   F   G   H   J   K   L   ’   \      ENTER
F4      SHIFT    Z   X   C   V   B   N   M   ,   .   -      SHIFT
F5      CTRL  LEFT RIGHT          SPACE             UP  DOWN   ALT
```

### 44.2.7 Spanish QL

```
F1      ESC  1   2   3   4   5   6   7   8   9   0   -   =   [   ]
F2      TAB    Q   W   E   R   T   Y   U   I   O   P   "   ^
F3      CAPS    A   S   D   F   G   H   J   K   L   Ñ   ;      ENTER
F4      SHIFT     Z   X   C   V   B   N   M   ,   .   <      SHIFT
F5      CTRL  LEFT RIGHT          SPACE             UP  DOWN   ALT
```

CHAPTER

FORTYFIVE

# A10 DESIGNING NEW CHARACTER SETS (FONTS)

A font is a character set used by the computer to display characters (normally letters and numbers) on screen. A wide range of fonts can be used, which allow you to make letters appear differently on screen for different programs.

## 45.1 A10.1 Fonts on the QL

On the QL, each window can have two fonts attached. There are actually two default fonts, the first of which defines the characters from 32 to 127 and the second of which defines the characters from 127 to 255.

The reason that CHR$(127) is defined twice is that the definition in the first font is the definition used for the copyright symbol, whereas the definition in the second font is that used if a character is undefined, such as CHR$(2).

Normally this is not noticeable and when you open a new window (scr_ or con_), the standard QL fonts are attached to that channel.

Minerva users can use the commands:

```
POKE_L !124!40, font1
```

and:

```
POKE_L !124!44, font2
```

to alter these default fonts and thereby attach user-defined fonts to every single window that is opened after this command.

SMSQ/E users can use the command CHAR_DEF to achieve a similar result.

## 45.2 A10.2 Changing Fonts in Programs

When a character is PRINTed, the QL looks up the code in the first font to find the binary definition of the character (see below). If the code is found in this first font then it is PRINTed, otherwise, the QL looks for the code in the second font and if found PRINTs this out.

If however the character is still not found, then the first character of the second font is printed out.

This therefore means that although substitute fonts need not have the same range as the QL standard fonts, it is important that a font contains all of the characters which will be used by a program!

On a QL, a font is stored in the following format:

| Offset | Value |
|---|---|
| $0 | Byte giving code of first character defined in the font |
| $1 | Byte giving the number of characters defined (minus 1) |
| $2…$A | Nine bytes defining the first character |
| $B… | Nine bytes for each subsequent character… |

Each character contained in the font is designed on a grid 8 pixels wide by 9 pixels high, and therefore the easiest way of calculating the nine bytes which make up each character is to design the character on a sheet of graph paper (if you do not have a font design program) and then each row must be converted from binary to decimal.

Each square in the grid which contains a 1 will be shown as a pixel in the current INK colour when PRINTed to the screen. For instance, this is the binary representation of the character k:

```
  |     BIT  7 6 5 4 3 2 1 0
--+-------------------------------
  |0         0 1 0 0 0 0 0 0 =  64
  |1         0 1 0 0 0 0 0 0 =  64
  |2         0 1 0 0 0 1 0 0 =  68
R |3         0 1 0 0 1 0 0 0 =  72
O |4         0 1 1 1 0 0 0 0 = 112
W |5         0 1 0 0 1 0 0 0 =  72
  |6         0 1 0 0 0 1 0 0 =  68
  |7         0 0 0 0 0 0 0 0 =   0
  |8         0 0 0 0 0 0 0 0 =   0
--+-------------------------------
```

Therefore a small program to set the character 'a' to the same as the character 'k' in channel #1 would be:

```
10 a=RESPR(11)
20 POKE a,97 : REMark 97=CODE('a') - first character in font
30 POKE a+1,0: REMark Only one character is being redefined
40 RESTORE
50 FOR i=0 TO 8:READ bit:POKE a+2+i,bit
60 DATA 64,64,68,72,112,72,68,0,0
70 CHAR_USE #1,a,0
```

Note that unfortunately no other characters will be printed correctly! This is because, when re-designing a font, it needs to include all of the characters which may be used in the window to which the font is attached.

When designing fonts for use on the QL, it is important that certain rules are followed to ensure that when characters are PRINTed on screen, they have the desired appearance in all screen modes. The QL converts a given character into the desired mode by doubling up various rows or columns (depending on the size set using the CSIZE command). If you intend to re-design letters, the rightmost two columns (bits 0 and 1) should always be left empty as these are ignored in current versions of the QL ROM.

The leftmost column (bit 7) is normally left blank to ensure a gap between adjacent characters, however, should you want to use this column also, you must be wary of the fact that pre-JS ROMs cannot display fonts correctly which use this column (bit 7) and so this should always be left blank unless you know that your font is going to be used specifically on later ROMs (see CSIZE).

In fact, if you are clever, you can use this bit 7 to obtain interesting effects on pre-JS ROMs, whilst still retaining the full 8 pixel wide characters (the effects vary depending on the ROM version and cannot be guaranteed!).

You should also be aware of the fact that an extra blank row is generally left between each line of characters. Toolkit II users can however prevent this by using CHAR_INC.

CHAPTER

FORTYSIX

# A11 MATHEMATICS

## 46.1 A11.1 Degrees and Radians

Normally an angle is expressed in so many degrees, for example 90 degrees is a right angle, 180 degrees is a straight line and 360 degrees is a circle. Unfortunately for those of us mortals out there, the QL works in a system of angles known as radians whereby a full circle measures 2*PI radians.

The diagrams make clear why 360 degrees and its multiples (720, 1080, ...) are identified with zero degrees.

```
degrees:  0        90            180           270        360
radians:  0       PI/2           PI          (3*PI)/2     2*PI

          |        |                            |          |
diagram:  |        |                            |          |
          +        +-----    -----+-----    ----+          +
```

A programming hint: If you are calculating angles and receive values for them where you have no guarantee that they are properly ranged, then use the remainder from the full circle angle instead. Since MOD finds the remainder for integers, you have simply to add a line such as:

```
degrees = degrees MOD 360
```

or:

```
degrees = MOD(degrees,360) (Math Package MOD)
```

for degrees, or:

```
radians = radians-2 *PI *INT(radians/2/PI)
```

for radians.

Note however that every function and command dealing with angles performs the same conversion internally or implicitly.

Just to make matters more confusing an angle in radians can be a maximum of 2*PI (which forms a circle). Thus PI is a straight line and PI/2 is a right angle. The relation between degrees and radians is:

```
radians = PI * degrees / 180
```

or:

```
radians = RAD(degrees)

degrees = 180 * radians / PI
```

or:

```
degrees = DEG(radians)
```

The QL does include the functions DEG and RAD which enable you to convert radians into degrees and degrees into radians (respectively). Unfortunately, however, all of the QL mathematical functions expect angles to be supplied in radians and therefore you must ensure that you are working in the correct system if you are to track down errors.

## 46.2 A11.2 Triangles and Trigonometrics

In order to explain some of the mathematical functions, you will need to envisage a right-angled triangle whose height is y and whose base length is x. Assuming that x is a line from the origin, the length of the line between the points (0,0) and (x,y) (the hypotenuse) is h. The angle formed between this line and the base line is theta radians. The maximum value of theta is 90 degrees (PI/2).

```
           /+ (x, y)
          / |
         /  |
        /   |
       /    |
     h/     | y
     /      |
    /       |
   / theta  |
  +---------+-----+
(0,0)  x  (x,0)
```

The following rules will give you an idea of the relationship between the various lengths and angles:

```
h^2 = x^2 + y^2
```

or:

```
h = SQRT(x*x + y*y)
```

or:

```
h = ABS(x,y): REMark Minvera only.
x = h * COS(theta)
y = h * SIN(theta)
theta = ATAN (y/x)
```

or:

```
theta = ATAN (x,y): REMark Minerva and SMS only.
theta = ACOT (x/y)
```

or:

```
theta = ACOT (y,x): REMark Minerva only.
y = x * TAN(theta)
x = y * COT(theta)
theta = ACOS (x/h)
theta = ASIN (y/h)
```

See the explanations of the keywords for details!

## 46.3 A11.3 Boolean Logic

The QL supports boolean logic which can be used in order to avoid lots of IF..END IF and SELect ON…END SELect structures. The idea behind boolean logic is that a statement is used to calculate an expression, which itself contains various logic operators and conditions. Please refer to the Operators section of this appendix for the order in which operators are calculated.

This can for example allow the following:

```
100 start_timer = 10: timer=start_timer: max_timer=100
110 REPeat loop
120   timer = timer + (timer < max_timer) - (timer + 1 - start_timer) *␣
↪(timer = max_timer)
130   PRINT timer
140 END REPeat loop
```

This program provides a timing counter, which counts from 10 up to 100 by one each pass of the loop and then re-starts at 10. Without boolean logic, this would have to be re-written:

```
100 start_timer = 10: timer = start_timer: max_timer=100
110 REPeat loop
120   timer = timer + 1
130   PRINT timer
140   IF timer = max_timer: timer = start_timer-1
150 END REPeat loop
```

This works because boolean logic works through an expression using the order of precedence (see the section on Operators). Whenever a comparison is found, this is evaluated to either 1 (true) or 0 (false) and then the rest of the expression evaluated. For example, x=y=0 will not, as some users may think, set both x and y to 0, but will set x to 1 if y=0 and x to 0 if y<>0. Therefore looking at line 120 in the first example, the following is carried out by the interpreter:

1. **timer =**… Note that we are assigning the final result to timer.
2. … **timer**…Stack current value of timer.
3. … **+(timer < max_timer)** …Calculate whether or not timer is less than max_timer. If true, add 1 to current value of timer, else add 0.
4. … **-(timer - start_timer)** …Stack the minus sign and then calculate the difference between the current value of timer and start_timer (this is the amount which will need to be deducted from timer to make it equal to start_timer).
5. … ***(timer=max_timer)** Calculate whether or not timer is equal to max_timer. If true, multiply the difference (from step 4) by 1, otherwise multiply it by 0.

6. Retrieve minus sign from stack and deduct value calculated in step 5 from the current value of timer. Assign current value to actual variable timer.

## 46.4 A11.4 Operators

Operators provide the QL (and any other computer) with a means of calculating an expression. An expression is always in the form:

term *[operator term]*

A list of available operators is set out below in order of precedence, that is to say that when the interpreter comes to calculate the value of an expression, which parts of the expression get calculated first. The order of precedence may be over-ridden by using parenthesis (brackets) - anything within a set of parenthesis gets calculated first, this is known as a sub-expression. For example, take the following expression:

x*y+(120-100-(50-20))

The interpreter will first of all calculate the value 50-20 which gives 30. Next, the interpreter needs to calculate 120-100-(30). As each operator is the same, this is carried out in an order from left to right, giving the value 20-30, in other words, -10.

This then leaves the interpreter with the expression x*y+(-10) to calculate. The multiplication operator takes precedence here, so the interpreter calculates the value x*y and then adds -10 to the result. This means for example, that if x is 20 and y is 5, this expression will return the value 90.

A term may be one of the following types:

- variable
- array element
- FuNctions
- strings
- values
- sub-expressions

A term may also be preceded by a Monadic Operator, which can be one of the following:-

- + this is a positive floating point. This can be omitted.
- - negate this floating point. eg. -x will if x=10 force this term to be equal to -10. However, if x=-10, this will force this term to be equal to 10.
- NOT perform logical NOT on this floating point - eg. NOT xwill, if x=0 force this term to be equal to 1. If however, x<>0, this term will be equal to 0.
- ~~ perform binary not on this integer - eg. ~~BIN('1001') will force this term to be equal to BIN('0110').

**NOTE 1**

On non-Minerva ROMs, monadic operators may only occur singly, which prevented expressions such as x=- NOT x. Minerva now allows this, for example, x%=-~~x% is the same as x%=x%+1 (this does not work with floating point numbers as ~~ can only work on integer values).

**NOTE 2**

On non-Minerva ROMs, negative values (eg. x=-1) are stored as a monadic positive operator, followed by a monadic negative operator. This no longer works on Minerva which stores negative numbers as merely a monadic negative operator.

Order of precedence of commands:

- + monadic operator - positive number eg: ++100 is the same as +100
- - monadic operator - negative number eg: +-100 is the same as -100
- & concatenates two strings together eg: 'Hello'&'World' => 'Hello World' (see Appendix 6.8)
- INSTR returns position of one string inside another (this is normally case independent, but see INSTR_CASE). Eg: 'world' INSTR 'Hello World' = 7
- ^ raise a floating point to the power of another floating point eg: 2^3=8
- * multiply a floating point by another floating point eg: 2*3=6
- / divide one floating point by another eg:10/5=2
- MOD return one integer modulus another integer, eg: 11 MOD 5=1
- DIV return the integer part of one integer divided by another eg:11 DIV 5=2
- + add two floating point numbers eg: 2+3=5
- - deduct a floating point from another eg: 2-5=-3
- > compare two values - is the first greater than the second? eg:x>2 for all values of x greater than 2
- >= compare two values - is the first greater than or equal to the second? eg:x>=2 for all values of x which are not less than 2
- = compare two values - is the first equal to the second? eg:'Hello'='HeLLo' is false
- == compare two values - is the first approximately equal to the second? (numeric values are approximately equal if they are equal to one part in 1E-7, whereas string variables are approximately equal if all of the characters are the same {ignoring case}). However, do note that nothing can ever be ==0, ie. x==0 will never be true (unless x is exactly equal to zero (ie. x=0). Instead, try x+1==1. Examples: 'Hello'=='HeLLo' is true '1.000000032'==1 is true
- <> compare two values - is the first value different from the second? eg:'Hello'<>'HeLLo' is true
- <= compare two values - is the first less than or equal to the second? eg:x<=2 for all values of x which are not greater than 2
- < compare two values - is the first less than the second? eg: x<2 for all values of x which are less than 2
- NOT monadic operator - logical not (see above)
- ~~ monadic operator - bitwise not (see above)
- AND logical and - are two floating point expressions true? eg:x=1 AND y=1 is true if both x and y are 1.
- && bitwise and - alter an integer value dependent upon a comparison bit by bit with the second integer value. eg:BIN('10001')&&BIN('111') returns BIN('00001')
- OR logical or - are either one or the other of two floating point expressions true? eg: x=1 OR y=1 is true if either x or y are 1.

- || bitwise or - alter an integer value dependent upon a comparison bit by bit with the second integer value. eg: BIN('10001')||BIN('111') returns BIN('10111')
- XOR logical exclusive or - are either one or the other of two floating point expressions true (but not both)? eg: x=1 XOR y=1 is true if either x or y are 1, but false if both are 1 or some other value.
- ^^ bitwise exclusive or - alter an integer value dependent upon a comparison bit by bit with the second integer value. eg:BIN('10001')^^BIN('111') returns BIN('10110')

## 46.5 A11.5 Hexadecimal and Binary Numbers

The original QL ROM could only work with decimal numbers which could cause some confusion when trying to work with machine code or using the bitwise operators to compare two values.

Toolkit II alleviated this somewhat with the introduction of the HEX, HEX$, BIN and BIN$ functions.

SMS and ST/QL Emulators (v1.27 of E-Init) have taken this one step further, by allowing hexadecimal and binary numbers to appear directly in SuperBASIC programs.

Hexadecimal numbers should be prefixed by the $ symbol, for example:

x=$4AFB is the same as x=19195

Binary numbers should be prefixed by the % symbol, for example:

x=%1010 is the same as x=10

**NOTE**

You need to process QREF_BIN to work with these new number types.

MasterBasic v1.46+ and Turbo v4.3+ can also cope with them.

## 46.6 A11.6 Integers

QLs have always been able to understand and use integer arithmetic, sometimes to speed up programs.

Minerva and SMS have extended the usefulness of the FOR and REPeat loops to allow them to use integer loop identifiers, which can be much quicker than using floating point identifiers (especially where the identifier is used to address an array).

Minerva has also introduced Integer Tokenisation which (when enabled) affects the way in which numbers are stored internally. This can both reduce memory requirements (and the size of a compiled program under Qliberator) as well as speed up programs. This can however cause problems - see QLOAD and POKE.

**NOTE**

Prior to v2.66 of SMS a=b%*c% would produce an overflow error where the result exceeded 32768.

Problems also existed where a=i%+j% and a<0 prior to v2.74.

There were also some other problems with integer arithmetic in versions prior to v2.31.

## 46.7 A11.7 Faster Mathematics

There are several ways of speeding up the QL's mathematics routines, such as using a faster processor (including some emulators and the THOR 21 Computer), SMSQ/E, Minerva or Lightning (a program by Digital Precision). You can even mix these together to get more improvement.

However, you can also use any maths co-processor which may be attached to your computer (see PROCESSOR) to speed up the routines substantially.

In order to do this, you will need to obtain the FPSAVE public domain toolkit together with an appropriate FPSP file (and also have a maths co-processor present - this is in-built on full 68040 and 68060 chips). You cannot use a maths co- processor with the original QL or with a Gold Card. If you have a QXL you will need to upgrade the 68040 chip to the full-blown model. However, the Atari TT and Falcon machines, the THOR 21 and 32 bit Amiga machines have either built in maths co-processors or sockets to take them.

The FPSAVE toolkit includes a set of functions which will replace the QL's native maths routines by faster ones which use the co-processor as well as another file containing the same functions prefixed with the letter F so that you can use both if you so wish. Unfortunately there are currently problems with using this toolkit on the Atari computers and you should use a copy of FPSAVE v1.17 at least to ensure that no other problems are encountered.

The functions which are speeded up by FPSAVE are:

ACOS, ACOT, ASIN, ATAN, COS, COT, EXP, LOG10, LN, SIN, SQRT, TAN

## 46.8 A11.8 Precision

The main problem with the QL's mathematics routines is the limited precision which is used by the native mathematics routines. Although the internal routines use a precision of at least 9 decimal places to calculate results, the Basic interpreter and PRINT commands will only accept figures six digits long for integers and seven digits long for floating point numbers. Any greater numbers are converted by PRINT and the interpreter to exponential notation, which means that the whole number is not stored.

To overcome this problem, you can either use Turbo or Supercharge to compile the program (these allow up to nine digits) or, if the number is to be stored within a BASIC program, place it in quote marks (as with the first example for the SCALE command).

CHAPTER

FORTYSEVEN

# A12 DEVICE DRIVERS

## 47.1 A12.1 Devices in General

This chapter informs you about the usage of the various devices available on the QL and its compatibles. The QL's operating system QDOS is what is known as device independent meaning that a program can be written to use any device without having to actually know its details (an exception to this rule is in the use of standard pipes). Programs should be written so that (at least) all of these devices can be accessed by the user as required.

Device Drivers are programs which usually create a connection between hardware devices and software, in that they install a QDOS device to interface from software to the hardware. For example a printer is obviously hardware but you do not have to POKE around in memory to get something printed, you can simply open a SER or PAR channel, dump your text to that channel and voila it appears on the paper. All communication with drivers must go through channels, whose name is very well chosen: they take data from the program and transport it to the device driver.

- The program opens channel and writes or reads data to/from that channel…
- The channel forwards the data (also instructions) to the device drivers…
- The driver is a kind of translator which understands the language of the hardware…
- A hardware interface translates computer codes into electrical signals…
- The hardware performs physical actions, eg. printing or reading from a floppy disk.

But devices can be used for all kind of connections, there are even general devices for communication between jobs (PIPE and HISTORY) and devices which interface in a special way to other devices (DEV and PTH). So the last two steps in the above figure are not obligatory, they can be different, ie. non-hardware.

Devices fall into two categories, directory devices (such as FLP) and non-directory devices. The latter may also be further sub-divided into window devices (devices which access the screen, such as scr) and other devices. All devices (other than window devices) accept the WIDTH command.

```
             all devices
               /      \
              /        \
directory devices    non-directory devices
                          /           \
                         /             \
              window devices       other devices
```

Further, if Toolkit II is present, or you are using a THOR XVI, certain commands will support default devices and also sub-directories on Level-2 directory devices.

Remember that device names can be in either upper or lower case, or even mixed case, that does not matter at all. Device independent programs should be fully configurable with regard to device names (eg. printer) and offer up to 42 characters for each device name. However, the ..._USE style commands and the DEV device help to overcome problems in this respect.

## 47.2 A12.2 Directory Device Drivers

Data in the form of files can be stored on various different directory device drivers, some of which allow data to be stored when the computer is switched off (such as Microdrives and disks) and others which lose their contents when the QL is reset or switched off (such as ramdisks).

Such media must be FORMATted prior to use in order to prepare them for use by the computer.

In order to speed up reading of these devices, unused parts of the computer's memory are set aside to act as slave blocks which store copies of as much of the contents of the device as possible. Then, when that same information is requested again, the computer need only check that the data held in the slave blocks is the same as on the device, and will then access the data from the slave blocks. This can however slow down the initial access times (see DEL_DEFB).

Each device contains a main directory which is made up of a copy of the 64-byte file header for each file which is (or has been) stored on the device. This main directory is then examined by commands such as DIR to produce a list of the files contained on that device. The file headers contain a host of different information about each file, including the name of the file, its type and the length of the file (see FGETH$).

When a file is accessed, various details (such as date-stamping creation and update dates) are updated.

The way in which information is stored on a directory device really depends upon the Level of device driver installed (see Sections A12.6 - A12.8 below).

Details of the types of Directory Devices follow:

### 47.2.1 Microdrive (MDV)

| | |
|---|---|
| Syntax | MDVn_file (QL ROM) or [MDVn_]file (Toolkit II only) |
| Location | QL ROM, THOR XVI |

This is the only standard directory device driver.

Microdrive cartridges are a continuous loop of video tape which store data in packets of 512 bytes (known as sectors). There is a theoretical maximum of 255 sectors on a Microdrive cartridge, although in practice the formatted number of sectors tends to be around 210 to 220.

The tape has to be searched serially in order to find the desired information which causes delay as the whole tape may have to be wound through in order to find the information. This is where slave blocks save a lot of time.

The standard QL supports two Microdrives (the slots to the right of the keyboard) into which Microdrive cartridges can be inserted on which the data is to be stored. However, up to a maximum of eight Microdrive ports can actually be recognised, if additional Microdrives are added by means of the Microdrive expansion port which is situated in the right hand side of the QL by the reset button.

The THOR XVI, and QL emulators do not possess any Microdrives, although the THOR does still recognise the device name, as it was originally envisaged that a separate Microdrive might be made to link up with the THOR in order to retain compatibility. The QL emulators will simply not find any Microdrive device like MDV1_, error -7 is produced. SMSQ/E also does not recognise the Microdrives.

Unfortunately, unless you have Toolkit II or Minerva, Microdrive files are not date-stamped with the creation and update dates.

Unless you have Toolkit II, the first syntax of this driver must be used, which specifies the number of the Microdrive port to access (n), which must be in the range 1…8, followed by the name of the file or device to access (file). If neither or these are specified, you are likely to receive the error 'Not Found' (-7), although whether or not the file has to be specified depends upon the command being executed.

If however, Toolkit II is present, the default devices are supported.

**Examples**

```
LOAD mdv1_boot
```

```
DIR mdv2_
```

Microdrives will allow you to create a file with a null name, which will not be revealed on a directory listing, but which will operate in much the same way as any other file. For example, the following two lines are both acceptable:

```
SAVE mdv1_Myprog_bas
SAVE mdv1_
```

Note that Microdrives do not possess Level-2 drivers and sub-directories are therefore not supported. For example creating a directory with:

```
MAKE_DIR mdv1_test_
```

will produce error -15 and leave the file test on mdv1_.

### 47.2.2 Floppy Disk (FLP)

| | |
|---|---|
| Syntax | FLPn_file or [FLPn_]file (Toolkit II only) |
| Location | Disk expansion boards, THOR XVI, QL Emulators |

This driver is for what are commonly known as floppy disks. These come in various sizes, ranging from 3" to 8", although the QL standard is now 3.5" double sided disks with either double (720k) or extra density (3.2 MB).

The amount of space on a disk depends on the number of tracks on the disk, the number of sides which can be used and the disk density. However, all of the drivers meet with a standard Sinclair format, ensuring compatibility between different manufacturers.

The drivers allow the same syntax as the MDV driver, although most disk expansion boards will support the second variant, as Toolkit II (or at least part of it) has become standard on disk interfaces.

**Examples**

```
MERGE flp1_Simple_bas
```

```
DATA_USE flp1_Quill
```

Some boards (such as the Gold Card, QXL, Atari Emulators and SMSQ/E) support Level-2 commands and therefore sub-directories. Other expansion boards can be fitted with these new drivers by updating a ROM chip. Level-2 drivers store the directory details in a separate file for simplicity. Unfortunately, the main directory is stored in a file with a null name which will automatically overwrite any earlier file with a null name. To see the main directory file, use:

```
COPY flp1_ TO scr
```

Sub-directories are stored in files with the name of the sub-directory. Such files have a file type of 255 and cannot be deleted or renamed until all of the files contained within that sub-directory have been removed.

Level-1 drivers supported files with null names in the same way as Microdrives (see above). On some old drivers FLP may be replaced by FDK.

### 47.2.3 RAMdisk (RAM)

| | |
|---|---|
| Syntax | RAMn_file or [RAMn_]file (Toolkit II only) |
| Location | QJump RAMPRT, Expansion Boards, THOR XVI, ST/QL Emulators, SMSQ/E, QXL, QPC, Amiga QDOS Emulator |

This driver is used to set up areas of memory which can be used in much the same way as a floppy disk. Anything stored in a RAMdisk is lost when the QL is reset or switched off.

There are actually two types of RAMdisks: a dynamic RAMdisk and a fixed RAMdisk.

A fixed Ram disk is allocated a size when the FORMAT command is used, and can contain anything between 3 sectors and the whole of free memory. Some fixed Ram disks (most notably the Qjump ram disk which is a standard) do not work on the Amiga- QDOS emulator - a slower public domain Ram disk which does work is supplied with the Emulator.

By contrast, a dynamic RAMdisk does not have a fixed size and is created when anything is written to it (do not use FORMAT) - it then expands and contracts to fit the size of the files contained in the RAMdisk.

Dynamic RAMdisks (optionally fixed) are supplied as standard on most QL systems. The RAMdisk drivers allow exactly the same syntax to the FLP driver, but the Miracle drivers (eg. Gold Card and Trump Card) support an extra syntax to format a RAMdisk to 255 sectors and copy a whole Microdrive cartridge into them, eg. FORMAT ram1_mdv1.

**Example**

```
WCOPY flp1_, ram2_
```

Level-2 drivers commands and sub-directories are also supported for the QJump RAMdisk driver, eg. on Gold Cards and ST/QL Emulators.

### 47.2.4 Hard Disk (WIN)

| | |
|---|---|
| Syntax | WINn_file or [WINn_]file |
| Location | Hard disk Interfaces, THOR XVI, QL Emulators |

This device driver allows you to access a hard disk drive (including removable hard disks). This operates a lot more quickly than a floppy disk (but not as quickly as a ram disk) and can store several megabytes of data.

Hard disks are built into nearly every system that can run a QL Emulator and are available as add-ons for a standard QL and AURORA system.

Please refer to the original manuals because the hard disk drivers all differ in FORMATting.

Accessing a WIN device from a program is just like accessing a FLP or RAM device.

### 47.2.5 QL ROMDisq (ROM)

| | |
|---|---|
| Syntax | ROM1_file or [ROM1_]file (Toolkit II only) |
| Location | QLROMDisq board |

This is a board which plugs into the QL's ROM Cartridge port and provides a fixed ram disk of either 2 or 8 Megabytes. It is similar to a RAM disk in that it is very quick when loading files, but it has three main differences:

- It retains its contents after the QL is switched off.
- You can only write data to it a limited number of times (100,000).
- It is fairly slow when you write files to it (with SAVE, SEXEC or SBYTES).

Because of these limits, this device is only really intended for storing files which will not change very often and are needed when the QL is started up (for example a new keyboard and language driver). A boot file stored on this device will be loaded when the QL is started up in preference to similar files on hard disks, floppy disks and microdrive cartridges.

The other main benefit of this device is that you can transfer whole set-ups across to another QL (instead of using lots of floppy disks or microdrives).

## 47.3 A12.3 Window Device Drivers

There are two types of window drivers, CON and SCR. The former (CON) is linked with a keyboard queue and can therefore accept input, as well as echoing any characters typed on screen. The latter (SCR) on the other hand is for output only to the screen.

Any channels opened using these screen drivers are known as windows, and may have a cursor attached to them. If a cursor is attached, then it will normally appear as a red blob on screen (the size and shape of the cursor may be redefined under Minerva v1.77+), and will flash when it is active (ie. when it will accept input).

When you press <CTRL><C>, QDOS cycles through all of the current cursors, allowing you to access different Jobs. If you are using Minerva in its two screen mode, then each channel is also attached to a screen, which means that if you open a channel on scr0, then all output to that channel will appear on scr0 whether or not that is the currently Displayed Screen (see MODE).

Also, when you press <CTRL><C> on Minerva, it will switch to the screen attached to the newly activated channel (unfortunately in current versions of Minerva, this does not quite work as expected, since if the active cursor is on the non-Displayed Screen when you press <CTRL><C> to move to a channel on the Displayed Screen, Minerva still switches screens, meaning that you are still not looking at the screen with the active cursor).

Whenever a new window is opened, it is opened with black paper and green ink. The specified pixel parameters are also rounded up to make them even (if necessary) to ensure that they can be correctly displayed in any screen mode. The smallest possible window is two pixels wide and one pixel high.

### 47.3.1 Console (CON)

| | |
|---|---|
| Syntax | CON[<size>][<position>][<buffer>] |
| Location | QL ROM |

This type of screen device is used for both output to the screen and reading the keyboard via a queue attached to that window. Depending on the command being executed, characters typed on the keyboard may be echoed on screen. This type of channel must be opened if you wish to use INPUT or INKEY$.

There are various problems with OPENing CONsole devices over the Network (see FSERVE).

When the computer is first started, there are three CONsole channels open, #0, #1 and #2; none of which should be CLOSEd or OPENed, this is especially true for #0!

When opening a channel, you can specify the size and position of the window and also the length of the type-ahead buffer attached to that window. These can have the following values:

<size>This sets the size of the window in pixel co-ordinates. It should be specified in the form:

[_WIDTH][xHEIGHT]

where WIDTH can have any value in the range 0…SCR_XLIM; and HEIGHT can have any value in the range 0…SCR_YLIM. The maximum values are however also dependent on <position>.

The default value for <size> is _448x200.

<position>This specifies the co-ordinates of the top left hand corner of the window and is in the form:

a[X][xY]

where X and Y can both be in the same range as WIDTH and HEIGHT (used in the <size>). However, both WIDTH+X and HEIGHT+Y must also be within the ranges, otherwise an 'Out of Range' error will be reported.

The default <position> is a32x16.

<buffer>This part of the device name specifies the size of the input buffer associated with the window, which is in the form _N bytes. This value affects how many characters can be stored in the channel's buffer before the keyboard has to be read again (this is known as the type-ahead buffer). Although this can have any value, a value of 128 bytes tends to be large enough for most tasks, and in fact this is the default.

Default Device:

```
CON_448x200a32x16_128
```

**Examples**

```
OPEN #3,con_200:     REMark Open channel #3 as CON_200x200a32x16_128
OPEN #3,con__10:     REMark Open channel #3 as CON_448x200a32x16_10
OPEN #3,cona12:      REMark Open channel #3 as CON_448x200a12x16_128
OPEN #3,conax20_50: REMark Open channel #3 as CON_448x200a32x20_50
```

The STE/QL emulator (QVME) and also other other hardware support much higher resolutions than 512x256, eg. QVME can go up to 1024x1024 pixels. However, programs should be written so that they still work with all other resolutions. This can be achieved by reading the possible screen size from system variables (the Pointer Environment must be used) and by not accessing screen memory directly.

### 47.3.2 Screen (SCR)

| | |
|---|---|
| Syntax | SCR[<size>][<position>] |
| Location | QL ROM |

This is very similar to the CONsole driver, except that SCR channels are for output to the screen only. No buffer size is required. Trying to read input from a SCR channel will give a 'Bad Parameter' (-15) error.

Please see the CON Window Driver.

Default Device:

```
SCR_448x200a32x16
```

## 47.4 A12.4 Other Device Drivers

In the following, LF is the line feed (or newline) character CHR$(10), CR is the carriage return character (13, $0D) and FF is the form feed character CHR$(12). In some applications <CTRL><Z> is used as an end of text character, CHR$(26).

Both parallel and serial ports are means for the QL to access other hardware in the outside world (such as printers, modems and scanners). Serial ports are so called because data is sent serially, one byte at a time. On the other hand, parallel ports allow several bytes to be sent at the same time and are therefore quicker.

Many printers are set up to accept parallel input and QL users may find that they need to purchase a serial to parallel converter (also known as a Centronics interface) in order to use a printer.

Each driver accepts various parameters which are used to match the output with the type expected by the device connected to the port. The main parameter deals with the parity of the byte to be sent. If no parity is specified, then all eight bits of the given byte will be sent, otherwise bit 7 of the byte will be altered according to the parity (this is best set according to what the hardware attached to the port requires).

You can also specify whether handshake is to be enabled, which tells the computer whether to wait for confirmation from the external hardware that the data has been received safely. If handshake is enabled, then if no acknowledgement is received, or the external hardware reports an error then the computer will try again.

Finally, you can specify whether the data is to be converted as it passes through the port. The standard code for ending a record or a line is CR, however, the QL is non-standard in that it uses the code LF, which therefore may need to be converted prior to transmitting.

Also, you may wish to send the character <CTRL><Z> as the last character in order to tell the external hardware that there is no more data.

### 47.4.1 Parallel Port (PAR)

| | |
|---|---|
| Syntax | PAR<new_line><trns><ff><buf> (THOR XVI) or |
| | PAR<port><translate><convert><eof> (SMSQ/E, ST Emulators, Super Gold Card) or |
| | PAR (AMIGA QDOS) |
| Location | THOR XVI, ST Emulators, SMSQ/E, Super Gold Card, Amiga QDOS Emulator |

Various QL implementations now come equipped with a parallel device driver for use with their parallel port. Parallel ports can be used for transmitting data only and are therefore normally used to connect parallel printers to the computer. Although there are various other expansion boards which also provide the standard QL with a PAR device, we do not currently have details of their syntax.

Note that even with SMSQ/E the PAR device does not exist on a Gold Card - there is no parallel printer port!!

The syntax of this device is quite complex, allowing different types of translations and buffers to be used. We shall therefore examine each variant in turn.

### 47.4.2 THOR XVI

The values of each part of the device name are as follows:

<new_line>This dictates how end of line (LFs) and end of text markers are to be treated. The following values are available:

- n - This converts LF to CR,LF and sends <CTRL><Z> at the end of the file. This is the default.
- c -This converts LF to CR and also sends <CTRL><Z> at the end of the file.
- r -This sends the text as it is - no conversions are carried out.
- z - This does not convert LF, but sends <CTRL><Z> at the end of the file.

The following table may be of use:

| EOL | EOF | Use This |
|---|---|---|
| CR,LF | CTRL-Z | n |
| CR | CTRL-Z | c |
| - | - | r |
| LF | CTRL-Z | z |

<trns> This tells the THOR XVI whether or not to use its translation tables (set with TRA). This can have the following values:

- t -Use the translation table. This is the default if <new_line> is specified but not raw.
- p -Do not use translation table. This is the default if <new_line> is not specified or is raw.

<ff> This says whether or not to send FF at the end of the file. The default depends on <new_line>. By default, a FF will be sent if <new_line> is set to n or c and the last character is not FF. The default can be overridden by setting <ff> to f which tells the THOR not to send FF unless of course there is already a FF at the end of the text!

<buf> This sets the size of the output buffer in bytes, and must be in the form _n, where n is the size of the buffer. If you add 'k' after the value of n, the value of n will be multiplied by 1024, for example _2K sets an output buffer of 2048 bytes.

The default is _127.

**Examples**

```
par_90k
```

Conversion of LF to CR, LF; translation table used; FF sent at end; buffer length 90 kilobytes.

```
parrt
```

No conversion; translation table used; no FF sent.

Note the coupling between the <New_line> and <trns> arguments. This means that par is equal to parnt, whereas parr is equal to parrp. The translation table used is the one set with TRA.

Default Device:

```
PARnt_128
```

### 47.4.3 ST Emulators, Super Gold Card AND SMSQ/E

These allow output through the parallel ports to be buffered dynamically, whereby a buffer is allocated up to all of the available free memory or (except on the Super Gold Card without SMSQ/E) can be set to a specific amount of space (thus allowing printing to continue in the background). Several channels may be open to one output port at any time, in which case the data is buffered and sent through the parallel port in the order in which the channels are opened.

Commands are implemented to allow you to set a specific output buffer or input buffer size (PAR_BUFF), as well as aborting output to a parallel port (PAR_ABORT) or clearing an output buffer (PARR_CLEAR).

The values of each part of the device name are as follows:

<port>This is provided for future compatibility. It represents the number of the parallel port to use. It can be either 1 or 2, although any attempt to use par2 is currently ignored and par1 used. The default is therefore 1.

<translate>This, like the THOR XVI's <trans> parameter specifies the type of translation to be carried out on the data. This can have the following values:

- d - No translation is performed.
- t - Translate according to the translate table. This is the default.
- <convert>This specifies how LF is to be treated. It can have the values:
  - c - This converts LF to CR.
  - r - No conversion, this is the default.
  - a -Insert CR,LF at end of line. Insert CR,FF at end of page (added to ST/QL drivers in Level D-05).
- <eof>This specifies how the end of the file is to be treated. It can be the default (do nothing) or have one of the following values:
  - f - Print FF at end of file

- z - Print CTRL-Z at end of file

**Example**

PAR1cz is the same as the THOR's PARn

Default Device:

```
PAR1tr
```

### 47.4.4 AMIGA-QDOS

This is the simplest form of parallel device driver, in that it does not accept any parameters (at least in v1.03 of the parallel driver). Any characters are sent straight through the Amiga's parallel port without being altered in any way.

### 47.4.5 Serial Ports (SER)

| | |
|---|---|
| Syntax | SER<prt><par><handshake><protocol>(QL only) or |
| | SER<prt><par><hand><translate><convert><eof> (ST Emulators, SMSQ/E) or |
| | SER<prt><par><bits><hds><bpso><bpsi><nl><trns><ff><buf> (THOR XVI only) or |
| Location | QL ROM, ST/QL, THOR XVI |

The QL, and THOR XVI are each equipped with two serial ports, marked SER1 and SER2 on the rear panel. Other implementations of the QL can in fact have access to up to four serial ports (even the standard QL can use additional serial ports built into SuperHermes for example).

If only one serial port is available (as on some STs), any attempt to use SER2 is treated as SER1.

The Amiga-QL emulator adopts a serial driver based on the JS version of the QL driver. It can be used to access either of the Amiga's two serial ports. It is however, unknown if current versions of the emulator's driver (v1.03) support CTRL Z.

Both ports on the British QL use non-standard British Telecom connectors and are actually wired up differently to each other (although they still use the same device driver). The other types of serial port tend to use standard 9-pin trapezium connectors.

On the standard QL, both ports conform with the RS-232-C standard, although the port marked SER1 is configured as a data communication equipment (DCE) port, which is normally used to drive printers; whereas the port marked SER2 is set up as a data terminal equipment (DTE) port, which is more suited towards accepting input from other devices (such as a modem). Refer to the QL User Guide manual, Concepts section, for further details of the hardware.

The rate at which data can be passed through the ports is known as the Baud rate which is set with the command BAUD from SuperBasic (or with a corresponding machine code trap call). This is supposed to be the number of bits per second, but due to the limitations of the QL's hardware, the rate of data transfer actually falls somewhat short of these rates (unless you have a THOR XVI, or use Minerva or SMSQ/E which has speeded up the rate of data transfer considerably).

Unfortunately, the 8049 IPC which controls input from both serial ports on the QL cannot handle different baud rates for the two ports. On the other hand, the THOR XVI does support different baud rates, although not via the BAUD command which sets the baud rate on both ports to the same. Hermes is a replacement for the QL's 8049 chip and allows different input baud rates on each of the two serial ports. Minerva allows different output baud rates on each port.

SMSQ/E and the ST Emulators allow you to set fully independent input and output BAUD rates on each port (although SuperHermes is still needed if this is to work on a standard QL).

Either port may be used for input or output (subject to hardware restrictions - see above), however, only one channel can be open to a serial port at a time, and if a channel is already open to the given port, the error 'In Use' will be reported.

The actual implementation of the SER device driver is dependent upon the machine. We shall therefore deal with each machine in turn.

### 47.4.6 Standard QL

This enables you to open a channel to either of the two serial ports. The action taken by the device driver depends both upon the ROM version being used and whether data is being input or output.

Note that input through the serial ports tends to be unreliable with baud rates in excess of 1200, and in any case, when receiving at 9600 baud, two stop bits must be issued by the transmitting device. Receiving at 19200 baud is not possible.

Unfortunately, problems in the 8049 mean that incoming data can be lost due to a delay in notification of the fact that the receive queue is full. Also, input channels can actually suffer from 'serial overrun' where some characters are held up in the 8049, and then released only when a new character is read from the serial port. This can sometimes happen with modems, making serial input unuseable.

There exists a replacement for the 8049 (called HERMES and its bigger brother SuperHERMES) which fixes these problems, as well as allowing separate baud rates for input and output channels, and which even supports different input rates on SER1 and SER2. This replacement also makes input at the higher baud rates much more reliable (including input at 19200 baud), without needing two stop bits at any rate.

The handling of both input and output is also dependent upon the ROM version being used:

#### (a) Output SERial devices

**Pre-JS ROMs**

If the C protocol has been chosen, then if the byte is a LF it is converted into a CR. Bit 7 of the byte is then adjusted to suit the parity and the byte then placed into the queue for the 8302 chip to read, deal with the handshaking and send through the channel. When the whole of the data has been sent, once the 8302 has emptied the queue, CTRL-Z is sent (if required).

Unfortunately, this meant that the protocol could just about be altered before the CTRL-Z had actually been sent, resulting in a failure by the QL to send any CTRL-Z's. This could happen for instance, if a series of small CTRL-Z files was sent to the serial channel and then the channel was re-opened as SERr. Another problem with the handling of CTRL-Z's was that the parity (if required) was not always correct on this final byte.

**JS and MG ROMs**

The serial driver followed the same pattern, except that if enabled by a TRA command (or the appropriate machine code call) the byte was translated according to the specified translation table after it had been adjusted to suit the parity (if required). This meant that bytes above CHR$(127) could not always be translated. The problems with CTRL-Z persisted.

**Minerva ROMs**

The serial driver is much improved, in that if the protocol is C, then LF is swapped with CR (and vice-versa). The byte is then translated according to the translation table (if required) and only then is it altered according to the parity setting.

The byte is then put into the queue to be sent to the 8302 and handshaking is then dealt with, leaving the 8302 to actually output the byte.

The problems with CTRL-Z have mainly been dealt with, although to overcome the problem of changing protocols, a channel structure linked to SERz or SERc cannot be discarded until all of the data in the transmit queue has been sent (meaning that the channel structure may not ever be discarded if handshaking forces the computer to keep trying to send the data). The main remaining problem is that in Minerva's two screen mode, characters may be lost on output.

### (b) Input SERial devices

**Pre-JS ROMs**

The 8302 deals with handshaking and then puts the byte which it has read into the receive queue. The device driver then reads the byte from the receive queue and checks the parity of the byte; reporting Xmit error if the check fails.

If the C protocol is chosen, then any CRs are converted into LFs and the byte returned to the user. Parity is completely ignored on CTRL-Z.

**JS and MG ROMs**

These both still suffer from CTRL-Z.

If enabled, a simple (one to one) translate is performed on the incoming byte as soon as it is fetched from the receive queue (see TRA). The parity is then altered as required, CRs converted into LFs (if necessary) and the byte passed onto the user.

**Minerva ROMs**

This checks the parity on CTRL-Z if required, along with the parity on any other data as soon as each byte is fetched from the receive queue. The byte is then translated (if necessary) according to the simple (one to one) translation table, CRs and LFs exchanged (unless protocol R chosen) and the byte then passed onto the user.

### (c) The Standard QL Device Driver

The parts of the device driver are made up of the following:

<prt> This specifies which serial port is to be used, and can be 1 or any higher number.

The default is ser1.

<par> This sets the type of parity to be used. The default is none, which allows all 8 bits of the characters to be sent. <par> may however be specified for one of the following values:

- e - Even
- o - Odd
- m - Mark
- s - Space

If a parity setting is used, then only seven bits of each code sent to the serial port are used, the last eighth bit is used to specify the parity.

If the parity is wrong when data is received through a port then the error 'Xmit Error' is reported.

<handshake>This specifies whether handshaking should be used. It may have the values:

- i - Ignore Handshaking
- h - Handshaking on. This is the default.

Handshaking is used to ensure that data is only sent through the serial port when the machine connected to the other end of the lead has sent a signal to say that it is ready to receive data.

<protocol>This specifies the type of conversion to be used. It may have one of the following values:

- r - No conversion carried out. This is the default.
- z - Use CTRL-Z for end of file flag.
- c - Convert LF to CR (or vice versa on input) and use CTRL-Z as end of file flag. Note: on Minerva, swap LF with CR on both input and output.

Default Device:

```
SER1hr
```

### 47.4.7 ST Emulators and SMSQ/E

These support a slightly enhanced variant of the device found in JS and MG ROM QLs (but with different bugs). Output through the serial ports can be buffered dynamically, whereby a buffer is allocated up to all of the available free memory or can be set to specific amount of space (thus allowing printing to continue in the background). Several channels may be open to one output port at any time, in which case the data is buffered and sent through the serial port in the order in which the channels are opened.

Commands are implemented to allow you to set a specific output buffer or input buffer size (SER_BUFF and SER_ROOM), as well as aborting output to a serial port (SER_ABORT) or clearing an output buffer (SER_CLEAR). Even the default handshaking can be set with SER_FLOW.

Serial ports may even be joined together to form a Network (SERNET).

When using SMSQ/E on standard QL serial ports hardware, there are several ways to improve the reliability:

- Use STX instead of SER to open output only ports.
- Use the command SER_PAUSE to alter the length of the stop bits on the serial ports.
- Fit Hermes (or SuperHERMES) - this is especially important for using higher BAUD rates and can improve the XON / XOFF protocol which can normally fail when trying to read data on the QL at over 2400 BAUD or trying to send data at over 4800 BAUD. Hermes is also needed to receive data at a different BAUD rate on each port and also at a different rate to the transmission rate.
- Change your serial to parallel converter - SMSQ/E is now so fast on the QL that some older converters no longer work correctly.

The SER device supports the various settings detailed on the following page. The default is ser1htr

<prt> This is the same as on the QL.

<par> This is also the same as on the QL.

<hand> This specifies whether or not to use handshaking, and if so which type is to be used. It can take the following values:

- h - Hardware Handshaking on - the default.
- i - Ignore handshaking

- x - XON/XOFF; no handshaking (see SER_ROOM).

Hardware Handshaking can only be used with a five-wire serial connector, as it uses one of the lines as a signal line to signify when the machine is ready to receive data.

XON/XOFF was added to ST/QL Emulators in Level D-00 drivers and also exists in SMSQ/E - it is software based handshaking and can be used with three-wire serial connectors. An XOFF character is sent to the other machine when there are only 32 characters left in the receive buffer (or other figure set with SER_ROOM), telling that other machine to stop sending data. Once there is room in the receive buffer for twice this number of characters an XON character is sent to the other machine which tells that machine to re- start data transmission.

SER_FLOW also affects this parameter.

<translate> This specifies the type of translation to be carried out on the data. This can have the following values:

- d - No translation is performed.
- t - Translate according to the translation table. This is the default.

The TRA command sets up translation tables.

<convert> This specifies how LF is to be treated. It can have the values:

- c - This converts LF to CR.
- r - No conversion, this is the default.
- a -Automatic insertion of CR,LF at end of line and CR,FF at end of page. This was added to ST/QL Drivers in Level D-05.

<eof> This specifies how the end of the file is to be treated. It can be the default (do nothing) or have one of the following values:

- f - Print FF at end of file
- z - Print CTRL-Z at end of file

### 47.4.8 THOR XVI

The serial ports provided on the THOR XVI use a much enhanced variant of the original JS device driver. The new serial device syntax is upwardly compatible with the original, ie. the old syntax described above is still accepted but additional parameters are allowed. The THOR also supports an enhanced translate table (see TRA).

The following parameters are now accepted by the device driver:

<prt> This is the same as on the standard QL driver.

<par> Again, as per the standard QL.

<bits> This digit sets the number of bits per byte to be sent. It can be 5, 6, 7 or 8. The default is 7 if parity is set, otherwise 8 for no parity.

<hds> This letter sets handshaking:

- h - on (default).
- i - ignore.
- x - XON/XOFF with handshaking.

- y - XON/XOFF without handshaking.

<bpso> This sets the current output baud rate and is specified as the number is preceded by a B. Valid parameters are: B75, B110, B134.5, B150, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200.

The system BAUD setting is the default. See BAUD.

<bpsi> This sets the input baud rate as above. A THOR XVI can send and receive data at different speeds. The default input baud rate is the current output baud rate.

<nl> This letter specifies how the end of line (EOL) and end of file (EOF) codes should be converted. This is the same as <new_line> in the THOR's PAR driver, except that the default here is r (raw).

<trns> This tells the THOR XVI whether or not to use its translation tables (set with TRA). This can have the following values:

- t - Use the translation table. This is the default if <nl> is not specified.
- p - Do not use translation table. This is the default if <nl> is specified.

<ff> This says whether or not to send FF at the end of the file. The default depends on <nl>. By default, a FF will be sent if <nl> is set to n or c and the last character is not FF. The default can be overridden by setting <ff> to f which tells the THOR not to send FF unless of course there is already a FF at the end of the text!

<buf> This sets the size of the output buffer in bytes, and must be in the form _n, where n is the size of the buffer. If you add 'k' after the value of n, the value of n will be multiplied by 1024, for example _2K sets an output buffer of 2048 bytes.

The default is _127.

**Example 1**

```
ser2exb75b1200cf
```

ser2 with even parity, send 7 bits per byte, XON/XOFF with handshake on, set output baud rate at 75 bps and input baud rate at 1200 bps, newline conversion to CR and use translate table, no form feed at end of file, use an output buffer of 127 bytes.

**Example 2**

```
ser7b1200
```

ser1 with no parity, send 7 bits per byte, normal handshake, both output and input baud rate set at 1200, no newline conversion (raw data) but use translate table, send form feed at end of file, use output buffer of 127 bytes.

Default Device:

```
ser18hrt_127
```

Note the coupling between the <nl> and <trns> arguments. This means that 'ser1' is equal to 'ser1rt', whereas 'ser1r' is equal to 'ser1rp'. The translation table used is the one set with TRA.

### 47.4.9 Serial Ports (SRX)

| | |
|---|---|
| Syntax | SRX<prt><par><hand><translate><convert><eof> |
| Location | ST Emulators, SMSQ/E |

This is a dedicated input only serial device, which has the same syntax as the ST Emulator's SER device.

Default Device:

```
SRX1htr
```

### 47.4.10 Serial Ports (STX)

| | |
|---|---|
| Syntax | STX<prt><par><hand><translate><convert><eof> |
| Location | ST Emulators, SMSQ/E |

This is a dedicated output only serial device, which has the same syntax as the ST Emulator's SER device.

It is recommended that if your program only needs to be able to send data out of the serial ports, this device is used, as this will enable other programs to open input devices (SRX) to the same serial port.

**NOTE**

On a standard QL, the same hardware is used for both serial ports, and therefore if you are using one port for input and one for output you should use the STX device on the output only port (instead of SER). If you use SER to open both ports then the speed of the input port will be unduly affected even though the other port is being used for output only. STX gets around this problem.

Default Device:

```
STX1htr
```

### 47.4.11 Printer Ports (PRT)

| | |
|---|---|
| Syntax | PRT |
| Location | Qjump RAMPRT, ST Emulators, SMSQ/E, QXL, Gold Card, Trump Card |

This is an unusual device driver which comes in two forms. However, in both forms, the idea is that a user will set up this device to point to the port which connects to his printer, so that a program merely needs to OPEN prt. In practice however, it is more advisable to allow the user to configure the program with the details of the port to be used for printing.

### 47.4.12 Qjump RAMPRT, Trump Card, QXL and Gold Cards

These allow the PRT device to be used to add buffers to serial and parallel ports (see PRT_USE).

### ST Emulators and SMSQ/E

On these implementations, the PRT device can be used to emulate either SER or PAR, but does not necessarily have a buffer attached. See PRT_USE.

### Memory Driver (MEM)

| | |
|---|---|
| Syntax | MEM_[adr1[_adr2]] (IODev) or<br>MEM[bufnr][_buflen{p\|t}] (DIY Toolkit) |
| Location | MEM device (DIY Toolkit Vol N), IODev (System) |

The memory device allows you to access RAM memory directly via a device. This is functionally the same as PEEKing the values with any of PEEK's available variants (PEEK$, PEEK_F etc), but the latter only allows you to access the memory of the local machine.

The MEM device on the other hand can be installed on a different machine connected via the Toolkit II fileserver, which allows you to use any device driver on a host machine through the n<nr>_ fileserver interface (see below).

Data can be read and written through a MEM device to memory with all commands and functions that work on files as well, so that memory becomes a file.

The DIY MEM device supports up to eight buffers of buflen bytes in size for data transfer between program and memory. A buffer is specified by bufnr, each buffer can be either temporary (t suffix to buffer length) or permanent (p). The file pointer needs to be explicitly set to the address location which you want to read from or write to.

IODev's MEM device has a much different syntax. The two modifiers adr1 and adr2 are numbers which indicate the start address (offset zero): 1024*adr1+adr2.

adr1 and adr2 are assumed zero if omitted.

**Example**

The classical demonstration for the MEM device is copying

the screen from one machine to another:

(1) IODev Variant

This can be easily done with:

```
SBYTES n2_mem_128,131072,32768
```

provided that the screen address is located at 131072 (128 * 1024 = 131072) on both machines and that both screens are 32k long. The above command copies the screen of the current machine to Network station number 2 (which must be running FSERVE).

(2) DIY Toolkit variant

This is defined differently and needs you to set the file pointer accordingly:

```
100 SBYTES_O ram1_q,131072,32768
110 OPEN#3,n2_mem_
120 GET#3\131072
130 SPL ram1_q TO #3
```

You will have noticed that both variants of MEM have an incompatible syntax. Fortunately however, it is still possible to write portable programs for both devices. Just use the most basic syntax; both MEMs will then behave identically and start at the absolute address zero.

This means that the above DIY Toolkit example will also work on the IODev variant (however ensure that the final underscore appears in line 110 to maintain DIY Toolkit compatability.

This example can be much improved by avoiding the need for a temporary file and extra code to check if SPL has finished (ignored here) by using FWRITE.

**NOTE**

MEM could have problems on Minerva pre v1.78.

**WARNING**

The use of the MEM device is not recommended because it uses absolute addresses on another machine. QDOS tends to move around all kinds of area of memory, so that even very sophisticated communication between the network partners cannot provide enough safety to avoid crashes.

Imagine the following (horror) scenario: Machine 1 tells machine 2 where its screen memory is located. Machine 2 starts to send its own screen to machine 1 but during the upload QDOS moves the screen on machine 1 to another location… BANG! The use of MEM must be declared as dirty or at least most dangerous. There are always alternatives which avoid MEM.

### 47.4.13 Network Drivers (NET)

| | |
|---|---|
| Syntax | NET<direction><station>(QL ROM) or |
| | NET<direction><station>_<buffer>(Toolkit II, THOR XVI) |
| Location | QL ROM, Toolkit II, THOR XVI |

These device drivers are explained separately in the Networks appendix.

### 47.4.14 Communication Drivers (PIPE)

| | |
|---|---|
| Syntax | PIPE_length(standard drivers) or |
| | PIPE[IDin]{X \| P \| T}IDout[_[length]][K](Minerva v1.97+) or |
| | PIPE_name[_length] (named drivers, SMS) |
| Location | QL ROM, named pipe drivers, SMS |

These are basically areas of memory which are set aside to act as communication queues. In theory, output data can be placed into the queue by a Job through one channel and the data can then be read by another Job (or the same Job) through another channel. The Job which is outputting data will be told when the pipe is full and will have to wait for something else to read some of the data before any more can be placed into the pipe.

Data is read out of a pipe in the order in which it is placed into it. This is known as First In First Out (FIFO).

Pipes can only be one way (either output or input). Any attempt to send data through an input pipe (or to read data from an output pipe) will cause a 'Bad Parameter' error. For compatibility reasons, you should open output pipes with OPEN_NEW and open input pipes with OPEN_IN.

A channel which is open to an input pipe cannot detect the end of data held within the pipe with the EOF command (unless the output channel has been closed) - instead, you will need to use the PEND or EOFW command to check if there is any more data waiting in the pipe. If you do not do this, then commands accessing the input pipe will merely wait around until they timeout (or wait indefinitely if the timeout is negative!).

More recently, the concept of Named Pipes has been introduced to QDOS which make the handling of pipes much easier, as you only need supply the name of the pipe to the input channel.

Again, we need to look at the various implementations of pipes:

### Standard QL ROM

**(1) Output Pipes**

It is easy to open an output pipe, with the syntax:

```
PIPE_length
```

where:

length Defines the length of the pipe, this is the number of bytes which can be stored in the queue at any one time. This cannot be extended at a later date (at least not very easily without losing all of the data).

length must be in the range 2…32767.

There is no default.

**(2) Input Pipes**

The problem comes when you try to link an input channel to this pipe. To do this, you need to open a channel to PIPE_0 with the channel ID of the first pipe in the machine code register D3.

Unfortunately there is no easy way of doing this in SuperBASIC, unless you have Minerva v1.82+ (see OPEN) or use a toolkit command such as QLINK which connects an existing channel to the given pipe.

**WARNING**

More than one input pipe may be connected to the same output pipe inadvertently, and you could even connect one input pipe to another. Both of these will eventually crash the system.

### Minerva ROM

This allows pipes to be created which are the same as on the standard QL ROM, except that length can have the letter K appended to multiply it by 1024. However, it is easier to link up input pipes to existing output pipes by using the extended OPEN commands implemented on Minerva v1.82+.

**Example**

Open a pipe between two programs, with a buffer of 10K

```
100 PCHAN=3
110 OPEN_NEW #PCHAN,pipe_10K
120 pipeID=PEEK_W(\48\PCHAN*40+2)
```

then in another program, having transferred the pipeID from the above program (by example using a temporary file):

```
130 OPEN_IN #5,pipe_,pipeID
```

However, a more flexible type of pipe has been implemented in Minerva v1.93+, with the syntax:

PIPE[IDin]{X | P | T}IDout[_[length]][K]

Pipes are identified by ID numbers (IDin) and (IDout), both of which can be any integer number in the range -32768 to 32767. The effect of omitting either ID numbers depends on the circumstances (see below).

In keeping with the other pipe drivers, length can be any integer between 0 ans 32767, appended by K if you want to multiply it by 1024. If omitted, it defaults to 0.

This sets up a system of pipes which are very similar to named pipes and channels can actually be opened to pipes which can both read from and output data to that pipe. The first channel to open a pipe to a particular IDin or IDout will need to specify the buffer length - any future channel which tries to specify a buffer length for the same pipe IDin or IDout will have no effect on the buffer.

The effect of the pipe depends on the values of IDin, IDout and whether the X, P or T parameter is specified.

- If IDin is omitted then the channel opened to the pipe will be write-only. IDin defaults to zero.
- If IDout is omitted or a negative number, and IDin is specified together with the X, P or T parameter then it will default to the same as IDin. - see (3) below. However, If IDout is omitted and the above paragraph does not apply, IDout is taken to be zero (or if IDout is specified to be zero) then the channel opened to the pipe will be read-only (you will need to specify X, P or T parameter if Minerva is to recognise IDout whether it is there or not).
- If both IDin and IDout are non-zero (or IDout was made to be the same as IDin under (2) above), then the channel opened to the pipe will read data from IDin and send data to IDout. If IDin and IDout are the same then this will form a circular queue.
- If both IDin and IDout are omitted and the X, P or T parameter is not specified, then you have created a standard QL pipe! If you specify the P or T parameter in this instance, see note 1 and note 2 below. PIPEX has no meaning!
- If a P parameter is specified, then this pipe will be marked as permanent and will retain its data even if no channels are open to it.
- A T parameter marks a pipe as temporary and can be used to remove a permanent pipe, eg: OPEN #3,'pipe1p2'Open a permanent input pipe (ID=1) and a permanent output pipe (ID=2). OPEN #3,'pipet2':CLOSE #3 Remove the pipe (ID=2) once all information has been read from it.
- An X parameter is used to merely separate IDin and IDout - this will create a temporary pipe which will mark the end of the data 'End of File' when the last channel which can output data to the specified pipe ID has been closed. When there are no channels at all left open to that pipe ID then any data in that pipe is lost and the memory released.

**Examples**

```
OPEN #3,pipe3x_100: REMark Open a read only pipe with a 100 byte buffer.
OPEN #3,pipex3_100: REMark Open the write only end of the above pipe.
```

Any easy way to transfer data between two programs:

From SuperBASIC enter the program:

```
100 OPEN #3,pipe1t3_300
110 REPeat wait_loop
120   INPUT #3,info$
130   IF info$='PROG 2 IS READY - SEND'
140     INPUT #3,datan,dummy$
150     EXIT wait_loop
160   END IF
170 END REPeat wait_loop
180 FOR i=1 TO datan
190   INPUT 'Enter Data Entry ';(i);': ';a$
200   PRINT #3,a$
210 END FOR i
220 REPeat end_WAIT
230   INPUT #3,a$
240   IF a$='THANKYOU':PRINT 'DATA SENT SUCCESSFULLY':EXIT end_WAIT
250 END REPeat end_WAIT
260 CLOSE #3
```

Use EX pipep to start a MultiBASIC and enter the program:

```
100 OPEN #3,pipe3t1_300
110 space=10
120 DIM rd$(space,100)
130 PRINT #3,'PROG 2 IS READY - SEND'
140 PRINT #3,space
150 PRINT #3,'DATA'
160 FOR i=1 TO space
170   INPUT #3,rd$(i)
180 END FOR i
190 PRINT #3,'THANKYOU'
200 CLOSE #3
```

Now RUN the program in the MultiBASIC, <CTRL><C> to SuperBASIC and enter RUN. Any data you enter into the SuperBASIC program will then be sent to the MultiBASIC program. Both programs can be RUN in either order!!

**NOTE 1**

PIPEP without any other parameters was implemented to automatically start up a MultiBasic, by using for example: EX pipep

see EW for details.

**NOTE 2**

PIPET opens a pipe similar to a NUL device - any attempt to read data from it will always report 'End of File', whereas any data sent to it will be thrown away.

**NOTE 3**

In v1.97 PIPEP and PIPET were the wrong way around when opened as an input pipe. You will therefore need to use a line such as:

```
IF VER$(1)='1.97': OPEN_IN #3, PIPEP2: ELSE: OPEN_IN #3, PIPET2
```

**NOTE 4**

It is recommended that in order to overcome problems with multitasking jobs trying to access the same pipe IDs inadvertantly, a Job should use its own Job number*100 plus the pipe ID number. This can be calculated with:

```
jobID = VER$(-1): JobNr = JobID - INT(JobID/65536)*65536
JobID = (PEEK_L(!!100) - PEEK_L(!!104)) DIV 4
```

### 47.4.15 Named Pipe Drivers, SMS

SMS, the ST/QL Emulators (Level D-00 onwards) and various other utilities implement named pipe drivers.

Named pipes solve many of the problems associated with QDOS's native pipes, in that there is no need to know the channel ID of the output pipe in order to open the input pipe.

Most versions will also allow you to open the input side before opening the output side, but oddly enough some non- standard versions will lose any data stored in a pipe if there is neither an output nor an input channel associated with it.

Input and output pipes can be closed in any order - information contained in them will not be lost, so you can close both the input and output pipes, but if there is still information contained in the pipe, you can then open a new input pipe to read this remaining data.

SMS v2.79 has further extended the concept of named pipes, allowing you to DELETE and DIR pipes.

```
DIR pipe
```

will list all named pipes which exist

```
DELETE pipe_name
```

will delete the specified pipe.

You can also:

```
VIEW pipe_name
```

In current drivers, a maximum of 15 or 16 named pipes can be open at any one time. The syntax of this driver is:

```
PIPE_name_length
```

for an output pipe, or:

```
PIPE_name
```

for an input pipe.

where:

name The name of the pipe (up to 32766 characters long), which must be in the standard name format (ie. the first character must be a letter or '_' symbol, with any other characters following).

There is no default.

length The length of the queue associated with the pipe (ie. how many characters can be stored in the pipe at a time). This must be in the range 2…32767. If a length of zero is given, this is taken to be an input pipe.

Default is 0.

**NOTE 1**

As there is no guarantee when you open an input channel to a pipe that it is empty (or contains only the information which you expect - for example another copy of your program may already be in use!!), it is normal for the first information to be sent by a program through a pipe to be some identification information (see the Minerva examples).

**Examples**

```
OPEN_NEW #3,PIPE_100
```

Open a standard output pipe which can hold up to 100 characters at a time.

```
OPEN_NEW #4,PIPE_xover_50
```

Open an output pipe named xover which can hold up to 50 characters at a time.

```
OPEN_IN #5,PIPE_xover
```

Open an input channel to the pipe xover.

```
OPEN_NEW #2,PIPE_quill_exp_100
```

Open an output pipe named quill_exp with a buffer for 100 characters.

```
PIPE_0
```

Open a general input channel to a pipe - see standard QL version above!

**NOTE 2**

Before Level D-06 of the ST/QL Drivers, PIPE_ or PIPE_0 could cause problems with TURBO compiled programs.

**WARNING**

When using SMSQ/E's named pipes, if you try to DELETE a pipe but a channel is OPEN to that pipe, then the error 'in use' is reported. However, when all channels to that pipe are CLOSEd, the pipe will immediately be DELETEd.

### 47.4.16 Communication Drivers (HISTORY)

| | |
|---|---|
| Syntax | HISTORY_name[_length] or HISTORY[_length] |
| Location | SMSQ/E |

The first syntax to this device creates a Public History Device - this is similar to the named pipes driver on the SMSQ/E except that it works as a Last In First Out (LIFO) device, so that information read from a HISTORY appears in the opposite order to which it was placed in the HISTORY and can be read from

any program. Also if a HISTORY device becomes full, the oldest message is thrown away. Messages are separated by NewLine characters.

Note that the name should not be a single character to allow for future improvements to this device.

The second syntax creates a Private History Device, which still stores messages so that they are fetched out of the History in the reverse order to how they were stored. However, no other program will be able to open a channel to this History and therefore it can only be read by the program which has opened it.

In both versions, if length is not specified, then it is presumed to be 1024 bytes.

As with Named Pipes, as from v2.79 of SMSQ/E, you can use DIR, VIEW and DELETE to get a directory of Public Histories, look at one of them and Delete them.

**Example**

Grab the name of the last file on a disk:

```
OPEN_NEW #4,HISTORY_FILE_10000
DIR #4,flp1_

VIEW HISTORY_file: REMark Just a quick look at the contents - it does not ␣
↪alter the contents of the HISTORY

INPUT #4,name$
CLOSE #4
DIR HISTORY
DELETE HISTORY_file
```

### 47.4.17 Nul Driver (NUL)

| | |
|---|---|
| Syntax | NUL or NULZ or NULF or NULL or NULP |
| Location | ST/QL Emulators and SMS |

A nul device is generally just an empty input only device that can consume anything put into it at great speed. It enables you to write software which can easily turn off its normal output by merely re-opening its output channel to a nul device should the user choose to do so.

All nul drivers are added by additional hardware and software. The standard device name is NUL, but there are also NULZ, NULF, NULL and NULP.

#### Input

The only real difference is if you try to read one of the nul drivers.

- NUL - This is an output only device, and returns bad parameter if you try to read information from it. Any attempt to read window information will return a zero parameter. Any attempt to read pointer information will return an invalid parameter.
- NULF - This emulates a null file - the EOF function will always be true on this channel. If you read the file header (with HGET for example) then a 14 byte header full of zeros is returned. Any attempt to read window information or pointer information has the same result as on NUL.
- NULL - This emulates a file filled with Line Feed characters CHR$(10). The file position can be set anywhere and the file header is 14 zero bytes. Any attempt to read window information or pointer information has the same result as on NUL.

- NULZ - This emulates a file full of zeros. You can set the file position to any value, but reading the header or data from the file will always return zeros. Any attempt to read window information or pointer information has the same result as on NUL.
- NULP - This will force the program to wait forever (or until any specified timeout has elapsed).

### Output

There is no difference when writing - all of the drivers just forget any data sent to them (eating it up at very high speeds).

## 47.5 A12.5 DIRECT SECTOR ACCESS

All standard directory device drivers (WIN, FLP, MDV and RAM) support direct sector access. This allow you to access the contents of a directory device without having to rely on the directory itself - it can therefore be used to rescue corrupt disks and even change the formatting of a disk (for example one utility uses this feature to squeeze extra room onto a normal Double Density floppy disk).

To use direct sector access, it is necessary to OPEN a channel to a special filename, in the form:

```
DRIVEn_*Dsd
```

Where:

DRIVEn_ This should be the name of the device followed by the drive number, for example FLP1_

*D This is the direct access identifier and must remain the same.

s This is a number which represents the length of a sector. s should be one of the following numbers:

- 0 = 128 bytes
- 1 = 256 bytes
- 2 = 512 bytes (DD and HD disks)
- 3 = 1024 bytes
- 4 = 2048 bytes (ED disks)

The value of 4 is only supported on Super Gold Cards. Values other than 2 are only supported on SMSQ/E and ST/QL Drivers from Level D-05 onwards.

d This is a letter in lower case which represents the density of the device, and should be:

- d - Double Density
- h - High Density
- e - Extra High Density

Once this file is OPENed, the file pointer is positioned at the start of the first sector of side 0 track 0 on the disk. Except on the ST/QL Drivers (pre D-04), no other file can be OPEN on the disk if this access is to work. For further details on how data is stored on a directory device, please refer to the documentation of the device driver.

**Example**

Read the name of a HD disk inserted in flp1_

```
100 OPEN_IN #3,'flp1_*D2h'
110 GET #3\1+ 0*256 + 0*2^16,sector$
120 CLOSE #3
130 PRINT 'The First Sector of the disk is;'sector$
140 PRINT 'The name of the disk appears in this sector - it is ';sector$(5 to␣
↪14)
```

**NOTE 1**

Some older floppy disk interfaces do not support this.

**NOTE 2**

MIDINET and SERNET include code to stop you using this facility over the Network to access protected files.

## 47.6 A12.6 Level-1 Device Drivers

These were the first Directory Device Drivers provided with the QL (and early QL floppy / hard disk controllers) and allowed the QL to access files on microdrives, ram disk, floppy disk and hard disk relatively easily. Hard disks introduced their own system of storing a directory of the files (and also some introduced their own non-standard sub-directory filing system).

DIR would produce a list of all files present on the device (normally) in the order in which they were created. You could even have files with no names (for example flp1_ ).

The main problem with these drives was when you had a hard disk (or even a floppy disk) containing hundreds of files, it could be very difficult to find the required file.

The drivers could also not read files which had been placed in sub-directories on disks created by Level-2 Device Drivers.

## 47.7 A12.7 Level-2 Device Drivers

Level-2 device drivers were first introduced for the ST/QL emulator but are now available on several emulators and QL expansion boards. These allow the user to define sub-directories of sets of files on certain directory devices.

Sub-directories are identified by grouping together all files which have the same prefix. The main directory will only recognise the name of the sub-directory and you will then need to look at the sub-directory to obtain details of that set of files. For example, DIR flp1_ may produce the following output:-

```
Example1
870/1440 sectors
boot
PSION->
```

This would show that the disk in flp1_ had the medium name 'Example1' (see FORMAT), had a maximum of 1440 sectors (720K) of which 870 remain unused, a boot file and a sub-directory called 'PSION' (see MAKE_DIR). You could then use:

```
DIR flp1_PSION
```

to produce the following:

```
Example1
870/1440 sectors
PSION_boot
PSION_Quill
```

This still provides the same information about the disk, but goes on to show that the sub-directory 'PSION' contains the files 'PSION_boot' and 'PSION_Quill'.

The use of sub-directories help to make the finding of files much more easy, especially on devices like hard disk drives where there could be several thousand file names to sort through.

## 47.8 A12.8 Level-3 Device Drivers

These device drivers provide all of the facilities of earlier device drivers (allowing you to create sub-directories on Hard Disks, RAM Disks and Floppy Disks), plus giving you the ability to read from and write to PC and Atari TOS disks.

The DIR command prints the type of the disk in the specified device.

Unfortunately, there are limited means of formatting PC and ATARI disks under the QL operating system and you have to use one of the various public domain utility programs to do so (or use the commands AFORMAT and IFORMAT if available).

If you try to write to an ATARI or PC format disk, then the filing system will look at the file name which you are using and if it is an invalid ATARI / PC filename (namely eight characters followed by a dot and a three letter extension) then a Not Found error may be reported. You are allowed to create sub-directories (up to four letters long) on an ATARI or PC disk with MAKE_DIR and also save filenames without an extension (up to a maximum of 8 characters still).

Oddly, however, if you try to precede the three letter filename extension by an underscore (as would normally be the case on the QL), this is not translated to a dot, instead, this underscore is counted as one of the 8 characters in the filename.

A slight inconsistency (possibly in the way in which PCs handle sub-directories) is that if you make a sub-directory with the command:

```
MAKE_DIR flp1_TEST
SAVE flp1_TEST.BAS
```

will not actually place this file in the sub-directory - compare SAVE flp1_TEST_TEST.BAS. You must also be aware that in keeping with PCs, you must create a sub-directory on DOS or TOS disks before you try to store a file in that sub-directory.

Until v2.52 of SMSQ/E, the filename needed to be in quote marks if it was to include a dot.

There is also a problem in that if you read some information from a DOS or TOS disk (for example with LOAD) and then remove that disk from the drive and alter it on another computer, replacing the disk in the original computer's disk drive (not having used another disk in the mean time), it is impossible to tell that the disk has been modified, so any further attempt to access that disk may render it unusable. If you must insist on doing this, use DEL_DEFB before trying to access the disk a second time.

One more difficulty that has been rectified from version 2.87 of SMSQ/E, is that on earlier versions you could not use EXEC or EXEC_W (or similar command) to execute a file from a DOS or TOS disk - this is because the file type will not be 1. You needed to copy the file from the DOS or TOS disk to RAM disk and then use commands such as:

```
OPEN #3,ram1_File_exe
HGET #3,length
HPUT #3,length,0,1,exdat
CLOSE #3
```

exdat will depend upon the file itself (it is the extra information which can be stored in a QDOS file header) - you will need to read this from the original QDOS version of the file when it was created.

You will also find that some programs will not be able to use PC formatted disks - for example the Psion programs (such as Quill and QL-Xchange) which will both add a three letter extension preceded by an underscore if one does not exist (such as _doc).

## 47.9 A12.9 Using Alien Format Disks

QL Emulators which run on non-QL based hardware normally have a means of creating a section of the hard-disk connected to a computer which can be used for storage of QL files.

QPC also includes commands to allow you to access CD-ROM drives (see CD_PLAY).

However, the problem comes when you need to try and read data from (or save data to) a floppy disk which is not in standard QDOS format.

There are several Public Domain and Commercial utilities which allow you to convert files from or to IBM or Atari Format disks into a QDOS format. Included amongst these utilities is the toolkit ATARIDOS (see IQCONVERT for example). Other good examples are the public domain IBMDISK program, the commercial program XOVER (by Digital Precision) and the shareware program MultiDISCOVER (by Dave Walker).

However, if you want really flexible access to such disks, then you will need an operating system which includes Level-3 Device Drivers (see above).

CHAPTER

# FORTYEIGHT

## A13 EXTENDED POINTER ENVIRONMENT

It would appear likely that future QDOS compatible operating systems will be written by Tony Tebby, the original designer of QDOS, and the author of the Extended Pointer Environment. The Extended Environment is supplied built-in with the Atari-QL Emulators, the QXL, and SMSQ/E, and is provided with various software packages for the Sinclair QL and other emulators (the only QDOS compatible computer it will not work on successfully is the Thor XVI).

When supplied with software, the user will receive a copy of the Pointer Interface (called ptr_gen), the Window Manager (called wman) and the Hotkey System II (called hot_rext). These system extensions are backwardly compatible and therefore you should only ever need to install the latest version of each package once to be able to run all software written with the Pointer Environment in mind (although see TH_FIX).

SMSQ/E comes complete with its own version of the Pointer Environment files built in and therefore these files should not be loaded into a computer with SMSQ/E - if the version of the Pointer Environment built into SMSQ/E is not recent enough for the software you are using (an error will be generated), you need to update your copy of SMSQ/E.

The Pointer Interface provides sensible control of the QL's multitasking abilities, ensuring that whenever part of a program's windows are covered by another program, that program is hidden and cannot try to access the screen (if you think of all of the programs' windows as tiles on the screen, only those tiles which are at the top of the pile can be accessed). It also provides you with a pointer which can be moved around the screen with either the cursor keys or a mouse in order to select different options or programs.

The Window Manager provides various utilities which enable programs to make use of the Pointer Interface, allowing them to generate menus which can be accessed by using the pointer, and which provide programs a similar feel - making it easier for a user new to the program to become accustomed to how to operate the program.

The Hotkey System II provides both a Hotkey System and a Thing System (both are independent of each other).

The Hotkey System allows you to set up various keys which (in combination with <ALT>) will provide direct access to different programs, as well as allowing you to stuff strings into the keyboard queue, pass information from one program to another and to recall the last line to have been typed. This facility is known as either ALTkeys or Hotkeys.

The Thing System is a means of providing QDOS with a list of named resources which can be accessed by different programs, which merely need to check if the resources they require are present.

Some general notes about writing programs which will work under the Pointer Environment appear in Section 4.

**NOTE 1**

If you have Hotkey System II installed, then you will need to use the command HOT_GO before any of the ALTkeys will work.

**NOTE 2**

Programs such as PIE and PEX affect the way in which the Pointer Environment works - see PIE_ON and PEON.

CHAPTER

# FORTYNINE

## A14 COERCION

The QL can coerce strings to transform them into numbers.

Unfortunately, the QL uses different rules on different ROMs and even different rules according to whether you are comparing two strings or whether you are assigning a value.

The following results have been obtained when testing a JM ROM and Minerva v1.93-v1.97:

```
PRINT '1'='1.':    REMark returns 1
PRINT '.'='0':     REMark returns 0
PRINT '.0'='0':    REMark returns 1
PRINT '1'=' 1':    REMark returns 0
PRINT '1'='1 ':    REMark returns 0
PRINT '.0'(1)='.': REMark returns 1

x='1.':   REMark is equivalent to x=1
x='.':    REMark gives 'error in expression'
x=' 1':   REMark is equivalent to x=1
x='.0':   REMark is equivalent to x=0
x='1 ':   REMark is equivalent to x=1
```

On SMSQ/E, the same results are obtained except that:

```
PRINT '1'='1.': REMark returns 0
PRINT '.0'='0': REMark returns 0
```

**NOTE**

x='.' is accepted on AH ROMs - becomes x=0

CHAPTER

FIFTY

# A15 MOUSE DRIVERS

A mouse in computing terms is a small box which can be pushed around the desk and as it moves, it is translated by the computer into cursor key movements and hence moves the cursor (or a pointer) on screen instead of using the keyboard.

Depending on the implementation, a mouse can make it very easy to use programs, providing a quick means of moving the pointer on screen.

The type of mouse which can be used and how you need to link it to the computer depends upon the QL implementation being used.

Many other devices have been created which send the same information as a PC mouse and should therefore work with drivers which support PC mice. This includes, trackerballs and bitpads.

## 50.1 A15.1 A Mouse for the Standard QL

There have been several types of mouse which have been produced over the years to be linked to a Sinclair QL. However, there are now only really three types of mouse commonly used with the QL and its various guises.

### 50.1.1 A15.1.1 Quanta Mouse (or QIMI Mouse)

This mouse is linked into a hardware interface which needs to be plugged into the computer - fitting involves opening up the QL case, carefully removing one of the microchips from its socket and plugging in the interface (plugging the microchip back into the top of the interface). A long lead is attached to the interface into which you plug the mouse.

The mouse needs to be an Atari-style 2 button mouse. Limited cursor key emulation is provided by holding down the left hand mouse button as the mouse is moved around.

This mouse will not work with some older versions of the Pointer Interface files (PTR_GEN and WMAN) - upgrade them if you notice a problem.

### 50.1.2 A15.1.2 AURORA Mouse Interface

The AURORA replacement motherboard, includes a socket into which a QIMI compatible mouse may be plugged - this emulates the QIMI Mouse Interface described above.

### 50.1.3 A15.1.3 Serial Mouse

This consists of a small wire connector which plugs into the QL's serial port and allows a standard PC serial mouse to be plugged in. This in itself will have no effect on the QL and you will also need to link in a serial mouse driver which will need to be set up for either a two-button mouse or a three button mouse (depending on which you have plugged into the serial port).

The main problem with using Serial Mice is that they tend to need the serial port set to BAUD 1200 which can be problematic if you need to use a printer for example, on the other serial port running at BAUD 9600. However, the serial mouse drivers can cope with this, generally suspending the mouse driver whenever the baud rate is altered, or if the other serial port is open (with a different Baud rate).

Another problem with Serial Mice is that they do not work very well with communications software (such as mail-box programs); unless the Modem (or mouse) is run through SuperHERMES. The problem is due to the original QL design of the serial ports - a link between the serial ports mean that if you move the mouse whilst using communications software, it will corrupt data.

You can also have problems of the serial ports holding onto the information sent by the mouse and then releasing it all at once (particularly with three-button mice).

Although a serial mouse can therefore be used on a standard QL, you should consider obtaining Hermes or SuperHermes which allow you to set independent BAUD rates for each port and thereby avoids this problem altogether (allowing you to still use the mouse whilst a channel is open to a modem for example). SuperHermes also provides additional serial ports which would allow you to use a Modem, Printer and Serial Mouse at the same time for example.

Another problem with serial mice is that the 3-button mice can be difficult to set up - some will automatically power up in 2-button mode unless you hold down a mouse key when you switch on the QL. Others have switches which force the mouse signals on a PC to generate straight vertical or horizontal movements. It is therefore recommended that you buy a suitable mouse from the supplier at the same time as buying the serial mouse driver!!

There are three Serial Mouse drivers available for the standard QL:

### SERMouse (by Albin Hessler Software)

This is provided with SMSQ/E for the Gold Card family.

It is ideally for use under the Pointer Environment, although you can use it to control the cursor as well if you prefer - see SERMCUR and SERMPTR. It will handle both 2 and 3- button mice.

If you want to be able to read the position of the pointer (as controlled by the mouse), you will need to use either EasyPTR or Qptr commands. There are however, several commands added to SuperBASIC to control the mouse - see SEMSPEED.

### DIY Toolkit Serial Mouse (Vol I)

This is a cardware version of a mouse driver, which comes with several versions, allowing use of 2 and 3-button mice and also versions which will only move a mouse pointer around the screen and ones which will also emulate the cursor keys and various buttons on the keyboard. You also have to load a version which is set up for the serial port which you intend to plug the serial mouse into.

Note that current versions do not currently move the Pointer in the Pointer Environment, although a commercial version of this driver is available (called ms_mus) which contains the same commands as SERMouse and can be used to control the Pointer, although this driver appears to be a little more selective over the serial mice which can be used with it.

Several commands are added to SuperBASIC to allow you to read the position of the mouse and control the mouse - see X_PTR% and PTR_ON.

### SuperHERMES

This includes a low speed serial interface into which a PC- style serial mouse can be plugged, in much the same way as the Albin Hessler SERMouse.

It emulates the QIMI Mouse Interface (see above).

## 50.2 A15.2 A Mouse for QPC / QXL

You cannot use a QL mouse driver with these emulators and will need instead to set up the system to load a DOS mouse driver before QPC or QXL is initiated. If the mouse does not have a PS/2 style mouse connector, you will also need to configure SMS so that it does not connect a serial port to the COM port to which the mouse is connected.

PS/2 style mice work with later versions of QPC (and all versions of QXL) without having to disable either serial port.

Having done this, the DOS mouse normally used with the PC can be used from within QPC and QXL to control programs written for the Pointer Environment.

Some early versions of SMSQ/E had problems if you disabled one of the serial ports (neither of them worked!) - you had to disable both serial ports for the mouse to work!!

## 50.3 A15.3 A Mouse for ATARIs

You cannot use a QL mouse driver with these emulators.

You can however use the Atari's mouse as soon as the file ATARI_xxx file is loaded which allows the Pointer Environment to work correctly with the mouse.

## 50.4 A15.4 A Mouse for Unix and Macintoshes

The QL emulators for these computers simply recognise the mouse which is normally used by the computer - do not try loading a QL mouse driver.

On the MacIntosh, you will need at least v2.1 of the Q- Emulator program if the mouse is to work with the Emulator.

You can however use the Atari's mouse as soon as the file ATARI_xxx file is loaded which allows the Pointer Environment to work correctly with the mouse.

## 50.5 A15.5 A Mouse for the Amiga

You cannot use a QL mouse driver with this computer.

Amiga QDOS has been able to use the Amiga's own mouse to control its software since v3.20. Various functions and commands have been added to SuperBASIC to control the mouse, as with the DIY Toolkit version (see PTR_ON).

CHAPTER

FIFTYONE

# A16 THE QL DISPLAY

The way in which the QL display is made up is fairly complex, and alters in different MODEs and on different resolutions. The extended display under SMSQ/E has also completely re-written the way in which the screen is addressed, causing some incompatibility problems.

The QL screen is in fact an area of the QL's memory which can be altered using PEEK and POKE (or similar commands) as well as the more usual display commands such as INK, PRINT, RECOL and INPUT. However, direct access to the screen should be avoided wherever possible, except via the machine code IOW.XTOP TRAP #3 routine (D0=$09).

In order to retain compatibility with older software, the Aurora motherboard, Q40/Q60 and QPC2 v3.00+ emulators all copy data stored at the standard QL screen address across to the correct display area. However, this in itself can lead to problems unless the computer is set up to start in 512x256 mode, since the software does not copy changes on the main screen (for example made with PRINT back to the original QL display area).

## 51.1 A16.1 The Screen Address

On a standard QL the screen is 32768 bytes long and stored in memory starting with the address 131072 ($20000 in Hexadecimal).

In dual screen mode, another QL screen is also stored in memory, normally at the address $28000 (in Hexadecimal) onwards.

However, in higher resolutions, this screen address has to move in order to make room for a larger screen size.

It is therefore imperative that programs and toolkit commands do not make assumptions about where the QL screen is stored - use SCR_BASE, SCREEN or similar functions to find the start address.

As the size of the screen alters, so does the amount of memory which the screen takes up - to find the number of bytes used to store a screen, use the formula:

screen_size = SCR_LLEN * SCR_YLIM

## 51.2 A16.2 The Screen Size

On a standard QL, the display normally supports 256x256 pixels in MODE 8 and 512x256 pixels in MODE 4.

However, if the QL implementation you are using allows you to alter the size of the QL's display (which can be anything up to 1600x1600 pixels), you can either configure the operating system to start up in a higher resolution or use a command such as SCR_SIZE.

Due to the differences in the possible displays, you should use the functions SCR_XLIM and SCR_YLIM to find out the maximum size of the screen which can be addressed by your program.

Another factor to be taken into account is the number of pixels which are used to contain the values of one pixel line of the display. On a standard QL this is 128 bytes and many programs assumed that this would never change. However, higher resolutions and extended colour drivers demand more storage space, so you should use SCR_LLEN to find out this number.

## 51.3 A16.3 On-Screen Colours

The QL screen is actually an area of memory which is specifically set aside to hold these details (the display memory). One of the QL's chips looks at this memory 50 times per second (60 times per second in the sK) and uses the values stored there to calculate the colour of the pixels which you see on the screen of your Monitor or TV. Emulators copy this screen to the area of memory used by the display card on the native machine.

The display memory starts at SCR_BASE which represents the top left hand corner of your Monitor's screen and the size of the memory in bytes is calculated by the formula:

SCR_YLIM * SCR_LLEN

As you will see from the information set out below, you can easily presume that if you know the number of pixels that a display size can show, you should be able to calculate SCR_LLEN (and vice-versa) and in fact some software does just this. However, this is not always so - some QL implementations use a fixed number of bytes to contain the displayed pixels (no matter what the screen resolution) and so you should use both SCR_LLEN and SCR_XLIM. See the examples below as to how programs should be written to take account of both of these factors.

The way in which the display memory is organised depends upon the screen mode being used, with more complex organisation methods for screen modes which display more colours.

Under SMSQ/E v2.98+. you are able to use either the Standard QL Colour Drivers, or the Enhanced Colour Drivers. If the latter is used, you need to specify for each program which colour scheme is to be used with the following commands:

| | |
|---|---|
| COLOUR_QL | use standard QL MODE 4 / MODE 8 colour definitions (this is the default scheme). |
| COLOUR_PAL | use 8 bit (256 colour) palette definition. |
| COLOUR_24 | use the 24 bit true colour definition. |
| COLOUR_NATIVE | use the native colour definition (dependent on the hardware itself). |

You can also specify that a different colour palette is to be used to represent each of the INK colours, using the commands:

| | |
|---|---|
| PALETTE_QL | Specify different palette for standard MODE 4/MODE 8 colours |
| PALETTE_8 | Specify different palette for 8 bit colours |

The MODE will always remain the same once a program is using the Enhanced Colour Drivers and the colour parameters expected by commands such as INK, PAPER, STRIP, BORDER and BLOCK will depend upon the following tables.

To use these tables, look up the hardware the program is to be used on and then find the colour you need (this will need to be specified as a PAL value, Native Colour Value or 24 Bit Colour Value depending on

which COLOUR_xx command has been used) - see COLOUR_PAL for an example of how to make a program adopt to the different hardware.

### 51.3.1 Standard QL Colour Drivers

#### MODE 4

This is one of the standard display modes supported on the QL and compatibles, with a lot of non-leisure software expecting this MODE - this is because it provides a minimum display area of 512 x 256 pixels.

On a standard QL colour scheme, every two bytes (a word) represent eight pixels on the Monitor's screen calculated by looking at the status of each of the corresponding eight binary bits in each byte. The first bit of the first byte is combined with the first bit of the second byte to represent the colour of the first pixel. The second bit of the first byte is combined with the second bit of the second byte to represent the colour of the second pixel.

For example, if the first two bytes stored at SCR_BASE are represented in binary as:

```
0 1 1 0 0 1 1 0   0 0 1 1 0 0 1 0
} |_______________}_|
}     2nd pixel   }
}_________________}
      1st pixel
```

The two bits are then placed side by side to create the colour combination, meaning that the first pixel is represented as 00 and the second pixel is represented as 10.

This provides us with the following colours:

| Bits | Colour |
|---|---|
| 00 | BLACK |
| 01 | RED |
| 10 | GREEN |
| 11 | WHITE |

Therefore in the above example, the first eight pixels of the display become:

00 10 11 01 00 10 11 00

which equates to the following colours:

BLACK, GREEN, WHITE, RED, BLACK, GREEN, WHITE, BLACK

**Example**

The following program will fill the screen with black and white vertical stripes:

```
100 MODE 4
110 FOR x=0 TO SCR_YLIM-1
120   FOR y=0 TO (SCR_XLIM-1)/4 STEP 2
130     POKE SCR_BASE+ (x*SCR_LLEN) + y,    BIN ('01010101')
135     POKE SCR_BASE+ (x*SCR_LLEN) + y+1,  BIN ('01010101')
140   END FOR y
150 END FOR x
```

### MODE 8

This was one of the standard display modes but is only fully supported on a limited number of QL implementations. A lot of leisure software expects this MODE - this is because it provides more colours and the possibility of flashing pixels on screen. However, if this mode is not available, fear not, since the programs will still run quite happily in other screen modes, although the screen may be a little different.

This mode provides a standard display area of 256 x 256 pixels.

On a standard QL colour scheme, every two bytes (a word) represent four pixels on the Monitor's screen calculated by looking at the status of each set of two corresponding binary bits in each byte. The first two bits of the first byte are combined with the first two bits of the second byte to represent the colour of the first pixel. The second two bits of the first byte are combined with the second two bits of the second byte to represent the colour of the second pixel.

For example, if the first two bytes stored at SCR_BASE are represented in binary as:

```
0 1 1 0 0 1 1 0   0 0 1 1 0 0 1 0
} } |__|__________}_}_|_|
} }    2nd pixel  } }
} }_______________} }
       1st pixel
```

The four bits are then placed side by side to define the pixel. The second bit specifies whether Flash is to be set (bit=1) - if flash is enabled by setting this bit, then this will affect all other pixels on that same line until another flash bit is set (disabling the Flash function).

The other three bits are combined to create the colour, meaning that the first pixel is represented as 0100 and the second pixel is represented as 1011.

This provides us with the following colours (excluding the flash bit which is represented here by an x):

| Bits | Colour |
|---|---|
| 0x00 | BLACK |
| 0x01 | BLUE |
| 0x10 | RED |
| 0x11 | MAGENTA |
| 1x00 | GREEN |
| 1x01 | CYAN |
| 1x10 | YELLOW |
| 1x11 | WHITE |

Therefore in the above example, the first four pixels of the display become:

| PIXEL | BITS | EFFECT |
|---|---|---|
| 0 | 0100 | BLACK (Turn Flash On at this Pixel) |
| 1 | 1011 | WHITE (Flashing) |
| 2 | 0100 | BLACK (Turn Flash Off after this Pixel) |
| 3 | 1010 | YELLOW |

**Example**

The following program will fill the screen with magenta and cyan flashing vertical stripes:

```
100 MODE 8
110 FOR x=0 TO SCR_YLIM-1
120   FOR y=0 TO (SCR_XLIM-1)/4 STEP 2
130     POKE SCR_BASE+ (x*SCR_LLEN) + y,    BIN ('01100110')
135     POKE SCR_BASE+ (x*SCR_LLEN) + y+1,  BIN ('11011101')
140   END FOR y
150 END FOR x
```

Note that only one in two cyan pixels flash, this is because the effect of each pass of the y loop is to set the following pixels:

| PIXEL | BITS | EFFECT |
|---|---|---|
| 0 | 0111 | MAGENTA (Turn Flash On at this Pixel) |
| 1 | 1001 | CYAN (Flashing) |
| 2 | 0111 | BLACK (Turn Flash Off after this Pixel) |
| 3 | 1001 | CYAN (Not Flashing) |

**SMSQ/E NOTES**

Under the Enhanced Colour Drivers, available under SMSQ/E v2.98+, COLOUR_QL can be used to make a program resemble the original MODE 4 or MODE 8, generating the same colours.

However, as explained in the description of the INK command, all 8 colours available to MODE 8 are actually available whether a program is attempting to run in MODE 4 or MODE 8. As a result, programs written for the original standard QL MODE 4 may show slight colour corruption.

It is possible to alter the set of 8 colours available if a different palette is specified with PALETTE_QL.

SMSQ/E can be forced to overcome any incompatibility problems by configuring it to load the Standard QL Colour Drivers; using DISP_COLOUR; or using PALETTE_QL.

### 51.3.2 Aurora Enhanced Colour Drivers

At present, a version of SMSQ/E which provides the Enhanced Colour Drivers for Aurora has not been released. The way in which these colour schemes are therefore used is subject to possible change.

Although this can be used for testing software, unfortunately, if an Enhanced Colour Mode is enabled on Aurora, the display is corrupted by pixels being split across the screen, effectively causing the screen to be repeated horizontally. Programs such as the Photon JPEG viewer overcome this by clearing the screen and only altering the display memory directly (not attempting to use any standard commands/ machine code operating system calls). See the examples below as to how this may be achieved.

The display mode may be changed directly by altering the value stored at address $18043 in memory (this is write only and cannot be read). The write-only Master Control Register at $18063 remains as on the standard QL for compatibility. Attempting to read the byte stored at $18043 will actually return the value of the Monitor Preset Register - see below.

### 51.3.3 The Master Control Register ($18063)

A write-only register where the following bits can be used:

| | | |
|---|---|---|
| Bit 0 | - | Blank Screen if set. |
| Bit 3 | - | Use MODE 4 if clear, MODE 8 if set. |
| Bit 7 | - | Display SCR0 if clear, SCR1 if set. Keep this bit clear if using non-standard QL display modes and resolutions. |

All other bits should be left clear.

As can be seen, Minerva's extended MODE calls alter this register and should be used where available.

### 51.3.4 The Enhanced Mode Control Register ($18043)

A write-only register where the following bits can be used:

Bits 0 & 1 - Control display resolution as per following table:

| Bit 1 | Bit 0 | Horizontal resolution |
|---|---|---|
| 0 | 0 | 512 pixels |
| 0 | 1 | 640 pixels |
| 1 | 0 | 768 pixels |
| 1 | 1 | 1024 pixels |

Bits 3 & 4 - Control colour mode as per following table:

| Bit 4 | Bit 3 | Mode |
|---|---|---|
| 0 | 0 | 4 Colour Mode (MODE 4) |
| 0 | 1 | 8 Colour Mode (MODE 8) |
| 1 | 0 | 16 Colour Mode |
| 1 | 1 | 256 Colour Mode |

Bit 7 - Control aspect ratio (which controls how the vertical resolution is calculated by reference to the horizontal resolution) as per following table:

| Bit 7 | Aspect Ratio |
|---|---|
| 0 | 2:1 (QL Style pixels); vertical res. = horizontal res. * 1/2 |
| 1 | 4:3 (Square pixels); vertical res. = horizontal res. * 3/4 |

All other bits should be left clear.

**IMPORTANT**

The actual resolution displayed will depend on the monitor preset, which can be read from the Monitor Preset Register (see below) and the mode selected (for reasons of limited high-resolution screen memory).

The resolution selected in the Enhanced Mode Control Register ($18043) in principle does NOT depend on the mode, except in MODE 8, where the resolution selected refers to MODE 4, but the number of pixels in one line is halved, as per the standard QL MODE 8 (this is to maintain compatibility), and by limit of the high-resolution screen memory.

Because the high-resolution screen memory is fixed at 240K, the resolutions in modes with more colours will be limited. In particular:

| | |
|---|---|
| MODE 4: | No limits (high-resolution screen memory is larger than maximum resolution of 1024 x 768 pixels). |
| 16 Colour Mode: | Maximum vertical resolution is limited to 480 lines. |
| 256 Colour Mode: | Horizontal resolution is limited to 512 pixels, and maximum vertical resolution is limited to 480 lines. |

Additional limits may apply depending on the monitor preset values.

The limiting logic is simple - if the resolution chosen is higher than a limit, the limit is used instead. Limits apply independently for x and y directions. The maximum x and y coordinates have to be adjusted according to these limits for every given resolution and monitor preset setting.

### 51.3.5 The Monitor Preset Register ($18043)

This is a read-only register where the following bits can be used:

| | | |
|---|---|---|
| Bit 0 | - | Interlace Enable Bit (IE) |
| Bit 2 (MT1) Bit 4 (MT0) } } | - | General Type of Monitor Selected |

The maximum vertical resolutions is calculated as per the following table (where NI means Not Interlaced and I means Interlaced):

| MT1 | MT0 | IE | Monitor type | Max. vert. resolution |
|---|---|---|---|---|
| 0 | 0 | 0 | QL standard | NI 288 lines |
| 0 | 0 | 1 | QL standard | I 576 lines |
| 0 | 1 | 0 | VGA | I 576 lines |
| 0 | 1 | 1 | VGA | I 768 lines |
| 1 | 0 | 0 | SVGA | NI 576 lines |
| 1 | 0 | 1 | SVGA | I 768 lines |
| 1 | 1 | 0 | Multisynch | I 768 lines |
| 1 | 1 | 1 | Multisynch diag. | 960 lines* |

* This is a special diagnostic mode which displays a 1024x960 interlaced picture on a multisynch monitor when 1024x768 is selected, hence displaying the contents of the whole high-res screen area. Whether the software will support this is optional - this combination of MT and IE bits is not used in normal operation.

### 51.3.6 16 Colour Mode

It is planned that under the Enhanced Colour Drivers available in SMSQ/E v2.98+, this mode will be available as MODE 8 and support up to 1024x480 resolution. It is not yet implemented and may be forced using the command:

| | | |
|---|---|---|
| POKE $18043,144 (144=%10010000) | - | 512 pixels x 480 pixels |
| POKE $18043,146 (146=%10010010) | - | 768 pixels x 480 pixels |
| POKE $18043,147 (147=%10010011) | - | 1024 pixels x 480 pixels |

(See above for details)

A different set of colours can be used by specifying a different palette with PALETTE_QL.

Actually writing to the screen directly causes some problems, since SCR_LLEN returns 256 bytes, although in actual fact, the screen is 512 bytes wide in this mode.

Under the Enhanced Colour Drivers, this mode uses a byte to store the colours of 2 pixels. Here, the four adjacent bits represent the same pixel.

The four bits are stored in the format **IRGB**, where:

- I is intensity
- G is Green
- R is Red
- B is Blue

It is uncertain how this will be implemented - However, the following table details the Native Values to be used when POKEing directly to the screen (in machine code for example) and the probable corresponding INK parameter to use to achieve that colour (**NOTE** this is not the same as the original QL colour scheme). Conversion of the values to binary gives a clue as to how this colour scheme works:

| Ink Value | Colour Name | IRGB Value Decimal | Value Hex | Value Binary |
|---|---|---|---|---|
| 0 | Black | 0 | $00 | 0000 |
| 1 | White | 15 | $0F | 1111 |
| 2 | Red | 12 | $0C | 1100 |
| 3 | Green | 10 | $0A | 1010 |
| 4 | Blue | 9 | $09 | 1001 |
| 5 | Magenta | 13 | $0D | 1101 |
| 6 | Yellow | 14 | $0E | 1110 |
| 7 | Cyan | 11 | $0B | 1011 |
| 10 | Dark Grey | 8 | $08 | 1000 |
| 11 | Grey | 7 | $07 | 0111 |
| 14 | Dark Red | 4 | $04 | 0100 |
| 17 | Green | 2 | $02 | 0010 |
| 19 | Blue | 1 | $01 | 0001 |
| ?? | Dark Magenta | 5 | $05 | 0101 |
| ?? | Dark Yellow | 6 | $06 | 0110 |
| ?? | Dark Cyan | 3 | $03 | 0011 |

**Example**

The following program for SMSQ/E will show the MODE 8 (16 colours) available on Aurora. Note the need to explicitly wipe the screen - this is because MODE would normally do this for you.

```
100 MODE 4
110 POKE $18043,144 : REMark force switch to MODE 8:COLOUR_PAL (512␣
↪resolution)
120 scr_offset=SCR_BASE(#1)
130 scr_len=512:REMark SCR_LLEN reports the wrong value in this mode
```

(continues on next page)

(continued from previous page)

```
140 :
150 REMark Blank out screen
160 col=0
170 FOR i%=0 TO 479
180   FOR j%=0 TO 508 STEP 4
190     POKE_L scr_offset+i%*scr_len+j%,col
200   END FOR j%
210 END FOR i%
220 :
230 REMark Draw Colours
240 yoff=20
250 FOR i=0 TO 1
260   xoff=0
270   FOR j=0 TO 15
280     col=j+j*2^4:REMark Set two pixels at a time.
290     scr_offset=yoff*scr_len+xoff+SCR_BASE(#1)
300     FOR a=0 TO 10
310       FOR b=0 TO 10
320           POKE scr_offset+a*scr_len+b,col
330       END FOR b
340     END FOR a
350     xoff=xoff+12
360   END FOR j
370 yoff=yoff+12
380 END FOR i
```

### 51.3.7 256 Colour Mode

It is planned that under the Enhanced Colour Drivers available in SMSQ/E v2.98+, this mode will be available as MODE 16. There is a fixed resolution available of 512x480 pixels. It is not yet implemented and may be forced using the command:

POKE $18043,154 (See above for details)

Here, every byte represents one pixel on the Monitor's screen, calculated by looking at the status of each of the binary bits in each byte.

Actually writing to the screen directly causes some problems, since SCR_LLEN returns 256 bytes, although in actual fact, the screen is 512 bytes wide in this mode.

The bits are combined to represent the amount of GREEN, RED and BLUE to be used for each pixel, in the format **GRBGRBGX**, where:

- G is Green
- R is Red
- B is Blue
- X is Red/Blue

The colours are hard to describe due to the range and therefore require experimentation to obtain the correct colours. However, the following table details the PAL colour which should be used as the INK parameter (

**NOTE**

this does not correspond with the original QL colour scheme!) and the corresponding Native Values to be used when POKEing directly to the screen (in machine code for example). It is not possible to list all 256 colours, therefore we have tried to list the most widely used ones (INK 0 to INK 63) grouped into the different colours. Conversion of the values to binary gives a clue as to how this colour scheme works:

| PAL Colour Value | Colour Name | GRBGRBGX Native Value (Decimal) | Native Value |
|---|---|---|---|
| 0 | Black | 0 | $00 |
| 1 | White | 255 | $FF |
| 8 | Dark Slate | 3 | $03 |
| 9 | Slate Grey | 28 | $1C |
| 10 | Dark Grey | 31 | $1F |
| 11 | Grey | 224 | $E0 |
| 12 | Light Grey | 227 | $E3 |
| 13 | Ash Grey | 252 | $FC |
| 58 | Cerise | 68 | $44 |
| 14 | Dark Red | 64 | $40 |
| 2 | Red | 73 | $49 |
| 63 | Deep Purple | 40 | $28 |
| 51 | Plum | 15 | $0F |
| 20 | Purple | 96 | $60 |
| 26 | Mauve | 100 | $64 |
| 57 | Faded Purple | 112 | $70 |
| 52 | Dusky Pink | 113 | $71 |
| 5 | Magenta | 109 | $6D |
| 21 | Shocking Pink | 105 | $69 |
| 45 | Dull Pink | 115 | $73 |
| 31 | Rose Pink | 239 | $EF |
| 39 | Pastel Rose | 253 | $FD |
| 27 | Peach | 235 | $EB |
| 50 | Midnight Blue | 7 | $07 |
| 19 | Dark Blue | 32 | $20 |
| 4 | Blue | 36 | $24 |
| 62 | Ultramarine | 48 | $30 |
| 49 | Dusky Blue | 23 | $17 |
| 44 | Steel Blue | 59 | $3B |
| 18 | Sea Blue | 160 | $A0 |
| 25 | Bright Blue | 164 | $A4 |
| 56 | Dull Blue | 168 | $A8 |
| 43 | Dull Cyan | 171 | $AB |
| 7 | Cyan | 182 | $B6 |
| 29 | Light Blue | 247 | $F7 |
| 30 | Sky Blue | 231 | $E7 |
| 48 | Dusky Green | 19 | $13 |
| 60 | Grass Green | 136 | $88 |
| 17 | Dark Green | 128 | $80 |
| 54 | Avocado | 198 | $C6 |
| 61 | Sea Green | 132 | $84 |

Table 4 – continued from previous page

| PAL Colour Value | Colour Name | GRBGRBGX Native Value (Decimal) | Native Value |
|---|---|---|---|
| 42 | Dull Green | 143 | $8F |
| 3 | Green | 146 | $92 |
| 23 | Lime Green | 210 | $D2 |
| 24 | Apple Green | 178 | $B2 |
| 55 | Dull Turquoise | 170 | $AA |
| 41 | Light Khaki | 199 | $C7 |
| 15 | Light Green | 243 | $F3 |
| 36 | Pastel Green | 254 | $FE |
| 46 | Brown | 11 | $0B |
| 59 | Tan | 80 | $50 |
| 6 | Yellow | 219 | $DB |
| 22 | Orange | 201 | $C9 |
| 16 | Mustard | 192 | $C0 |
| 47 | Khaki | 27 | $1B |
| 53 | Buff | 197 | $C5 |
| 40 | Brick | 87 | $57 |
| 33 | Beige | 249 | $F9 |
| 28 | Light Yellow | 251 | $FB |

It is unknown how PAL colours 32, 34, 35, 37 and 38 will be mapped as these relate to the same values as PAL colours 31, 33, 36, 13 and 13 respectively.

The remainder of the colours are mapped as **grbgrbgx** (we would welcome names for each of these colours):

| PAL Colour Value | GRBGRBGX Native Value (Decimal) | Native Value (Hex) | Native Value (Binary) |
|---|---|---|---|
| 64 | 4 | $04 | 00000100 |
| 65 | 1 | $01 | 00000001 |
| 66 | 5 | $05 | 00000101 |
| 67 | 33 | $21 | 00100001 |
| 68 | 37 | $25 | 00100101 |
| 69 | 8 | $08 | 00001000 |
| 70 | 12 | $0C | 00001100 |
| 71 | 44 | $2C | 00101100 |
| 72 | 9 | $09 | 00001001 |
| 73 | 13 | $0D | 00001101 |
| 74 | 41 | $29 | 00101001 |
| 75 | 45 | $2D | 00101101 |
| 76 | 65 | $41 | 01000001 |
| 77 | 69 | $45 | 01000101 |
| 78 | 97 | $61 | 01100001 |
| 79 | 101 | $65 | 01100101 |
| 80 | 72 | $48 | 01001000 |
| 81 | 76 | $4C | 01001100 |
| 82 | 104 | $68 | 01101000 |

continues on next page

Table 5 – continued from previous page

| PAL Colour Value | GRBGRBGX Native Value (Decimal) | Native Value (Hex) | Native Value (Binary) |
|---|---|---|---|
| 83 | 108 | $6C | 01101100 |
| 84 | 77 | $4D | 01001101 |
| 85 | 2 | $02 | 00000010 |
| 86 | 6 | $06 | 00000110 |
| 87 | 34 | $22 | 00100010 |
| 88 | 38 | $26 | 00100110 |
| 89 | 35 | $23 | 00100011 |
| 90 | 39 | $27 | 00100111 |
| 91 | 10 | $0A | 00001010 |
| 92 | 14 | $0E | 00001110 |
| 93 | 42 | $2A | 00101010 |
| 94 | 46 | $2E | 00101110 |
| 95 | 43 | $2B | 00101011 |
| 96 | 47 | $2F | 00101111 |
| 97 | 66 | $42 | 01000010 |
| 98 | 70 | $46 | 01000110 |
| 99 | 98 | $62 | 01100010 |
| 100 | 102 | $66 | 01100110 |
| 101 | 67 | $43 | 01000011 |
| 102 | 71 | $47 | 01000111 |
| 103 | 99 | $63 | 01100011 |
| 104 | 103 | $67 | 01100111 |
| 105 | 74 | $4A | 01001010 |
| 106 | 78 | $4E | 01001110 |
| 107 | 106 | $6A | 01101010 |
| 108 | 110 | $6E | 01101110 |
| 109 | 75 | $4B | 01001011 |
| 110 | 79 | $4F | 01001111 |
| 111 | 107 | $6B | 01101011 |
| 112 | 95 | $5F | 01011111 |
| 113 | 16 | $10 | 00010000 |
| 114 | 20 | $14 | 00010100 |
| 115 | 52 | $34 | 00110100 |
| 116 | 17 | $11 | 00010001 |
| 117 | 21 | $15 | 00010101 |
| 118 | 49 | $31 | 00110001 |
| 119 | 53 | $35 | 00110101 |
| 120 | 24 | $18 | 00011000 |
| 121 | 56 | $38 | 00111000 |
| 122 | 60 | $3C | 00111100 |
| 123 | 25 | $19 | 00011001 |
| 124 | 29 | $1D | 00011101 |
| 125 | 57 | $39 | 00111001 |
| 126 | 61 | $3D | 00111101 |
| 127 | 84 | $54 | 01010100 |
| 128 | 116 | $74 | 01110100 |
| 129 | 81 | $51 | 01010001 |

continues on next page

Table 5 – continued from previous page

| PAL Colour Value | GRBGRBGX Native Value (Decimal) | Native Value (Hex) | Native Value (Binary) |
|---|---|---|---|
| 130 | 85 | $55 | 01010101 |
| 131 | 117 | $75 | 01110101 |
| 132 | 88 | $58 | 01011000 |
| 133 | 92 | $5C | 01011100 |
| 134 | 120 | $78 | 01111000 |
| 135 | 124 | $7C | 01111100 |
| 136 | 89 | $59 | 01011001 |
| 137 | 93 | $5D | 01011101 |
| 138 | 121 | $79 | 01111001 |
| 139 | 125 | $7D | 01111101 |
| 140 | 18 | $12 | 00010010 |
| 141 | 22 | $16 | 00010110 |
| 142 | 50 | $32 | 00110010 |
| 143 | 54 | $36 | 00110110 |
| 144 | 51 | $33 | 00110011 |
| 145 | 55 | $37 | 00110111 |
| 146 | 26 | $1A | 00011010 |
| 147 | 30 | $1E | 00011110 |
| 148 | 58 | $3A | 00111010 |
| 149 | 62 | $3E | 00111110 |
| 150 | 63 | $3F | 00111111 |
| 151 | 82 | $52 | 01010010 |
| 152 | 86 | $56 | 01010110 |
| 153 | 114 | $72 | 01110010 |
| 154 | 118 | $76 | 01110110 |
| 155 | 83 | $53 | 01010011 |
| 156 | 119 | $77 | 01110111 |
| 157 | 90 | $5A | 01011010 |
| 158 | 94 | $5E | 01011110 |
| 159 | 122 | $7A | 01111010 |
| 160 | 126 | $7E | 01111110 |
| 161 | 91 | $5B | 01011011 |
| 162 | 95 | $5F | 01011111 |
| 163 | 123 | $7B | 01111011 |
| 164 | 127 | $7F | 01111111 |
| 165 | 129 | $81 | 10000001 |
| 166 | 133 | $85 | 10000101 |
| 167 | 161 | $A1 | 10100001 |
| 168 | 165 | $A5 | 10100101 |
| 169 | 140 | $8C | 10001100 |
| 170 | 172 | $AC | 10101100 |
| 171 | 137 | $89 | 10001001 |
| 172 | 141 | $8D | 10001101 |
| 173 | 169 | $A9 | 10101001 |
| 174 | 173 | $AD | 10101101 |
| 175 | 196 | $C4 | 11000100 |
| 176 | 228 | $E4 | 11100100 |

continues on next page

Table 5 – continued from previous page

| PAL Colour Value | GRBGRBGX Native Value (Decimal) | Native Value (Hex) | Native Value (Binary) |
|---|---|---|---|
| 177 | 193 | $C1 | 11000001 |
| 178 | 225 | $E1 | 11100001 |
| 179 | 229 | $E5 | 11100101 |
| 180 | 200 | $C8 | 11001000 |
| 181 | 204 | $CC | 11001100 |
| 182 | 232 | $E8 | 11101000 |
| 183 | 236 | $EC | 11101100 |
| 184 | 205 | $CD | 11001101 |
| 185 | 233 | $E9 | 11101001 |
| 186 | 237 | $ED | 11101101 |
| 187 | 130 | $82 | 10000010 |
| 188 | 134 | $86 | 10000110 |
| 189 | 162 | $A2 | 10100010 |
| 190 | 166 | $A6 | 10100110 |
| 191 | 131 | $83 | 10000011 |
| 192 | 135 | $87 | 10000111 |
| 193 | 163 | $A3 | 10100011 |
| 194 | 167 | $A7 | 10100111 |
| 195 | 138 | $8A | 10001010 |
| 196 | 142 | $8E | 10001110 |
| 197 | 174 | $AE | 10101110 |
| 198 | 139 | $8B | 10001011 |
| 199 | 175 | $AF | 10101111 |
| 200 | 194 | $C2 | 11000010 |
| 201 | 226 | $E2 | 11100010 |
| 202 | 230 | $E6 | 11100110 |
| 203 | 195 | $C3 | 11000011 |
| 204 | 202 | $CA | 11001010 |
| 205 | 206 | $CE | 11001110 |
| 206 | 234 | $EA | 11101010 |
| 207 | 238 | $EE | 11101110 |
| 208 | 203 | $CB | 11001011 |
| 209 | 207 | $CF | 11001111 |
| 210 | 144 | $90 | 10010000 |
| 211 | 148 | $94 | 10010100 |
| 212 | 176 | $B0 | 10110000 |
| 213 | 180 | $B4 | 10110100 |
| 214 | 145 | $91 | 10010001 |
| 215 | 149 | $95 | 10010101 |
| 216 | 177 | $B1 | 10110001 |
| 217 | 181 | $B5 | 10110101 |
| 218 | 152 | $98 | 10011000 |
| 219 | 156 | $9C | 10011100 |
| 220 | 184 | $B8 | 10111000 |
| 221 | 188 | $BC | 10111100 |
| 222 | 153 | $99 | 10011001 |
| 223 | 157 | $9D | 10011101 |

continues on next page

Table 5 – continued from previous page

| PAL Colour Value | GRBGRBGX Native Value (Decimal) | Native Value (Hex) | Native Value (Binary) |
|---|---|---|---|
| 224 | 185 | $B9 | 10111001 |
| 225 | 189 | $BD | 10111101 |
| 226 | 208 | $D0 | 11010000 |
| 227 | 212 | $D4 | 11010100 |
| 228 | 240 | $F0 | 11110000 |
| 229 | 244 | $F4 | 11110100 |
| 230 | 209 | $D1 | 11010001 |
| 231 | 213 | $D5 | 11010101 |
| 232 | 241 | $F1 | 11110001 |
| 233 | 245 | $F5 | 11110101 |
| 234 | 220 | $DC | 11011100 |
| 235 | 248 | $F8 | 11111000 |
| 236 | 221 | $DD | 11011101 |
| 237 | 150 | $96 | 10010110 |
| 238 | 151 | $97 | 10010111 |
| 239 | 179 | $B3 | 10110011 |
| 240 | 154 | $9A | 10011010 |
| 241 | 158 | $9E | 10011110 |
| 242 | 186 | $BA | 10111010 |
| 243 | 190 | $BE | 10111110 |
| 244 | 155 | $9B | 10011011 |
| 245 | 159 | $9F | 10011111 |
| 246 | 187 | $BB | 10111011 |
| 247 | 191 | $BF | 10111111 |
| 248 | 214 | $D6 | 11010110 |
| 249 | 242 | $F2 | 11110010 |
| 250 | 246 | $F6 | 11110110 |
| 251 | 211 | $D3 | 11010011 |
| 252 | 215 | $D7 | 11010111 |
| 253 | 222 | $DE | 11011110 |
| 254 | 250 | $FA | 11111010 |
| 255 | 223 | $DF | 11011111 |

**Example**

The following program for SMSQ/E will show the full range of colours available on Aurora. Note the need to explicitly wipe the screen - this is because MODE would normally do this for you.

```
100 MODE 4
110 POKE $18043,156 : REMark force switch to MODE 256:COLOUR_PAL
120 :
130 scr_offset=SCR_BASE(#1)
140 scr_len=512:REMark SCR_LLEN returns the wrong figure in this mode
150 :
160 REMark Blank out screen
170 col=0
180 FOR i%=0 TO 479
```

(continues on next page)

(continued from previous page)

```
190   FOR j%=0 TO 508 STEP 4
200     POKE_L scr_offset+i%*scr_len+j%,col
210   END FOR j%
220 END FOR i%
230 :
240 REMark Draw Colours
250 yoff=20
260 FOR i=0 TO 15
270   xoff=0
280   FOR j=0 TO 15
290     col=i*16+j
300     scr_offset=yoff*scr_len+xoff+SCR_BASE(#1)
310     FOR a=0 TO 10
320       FOR b=0 TO 10
330         POKE scr_offset+a*scr_len+b,col
340       END FOR b
350     END FOR a
360     xoff=xoff+12
370   END FOR j
380 yoff=yoff+12
390 END FOR i
```

### 51.3.8 QPC/QXL Enhanced Colour Drivers

SMSQ/E v2.98+ provides various colour modes for QPC2 and the QXL card. You can configure SMSQ/E to start with either the Standard QL Colour Drivers or the Enhanced Colour Drivers. If the Enhanced Colour Drivers are loaded, RMODE will return 32.

The Enhanced Colour Drivers support a QL 8 colour mode (selected with COLOUR_QL), a PAL Colour Mode providing 256 colours (selected with COLOUR_PAL), a Native Colour Mode providing 65536 colours (select with COLOUR_NATIVE) and a 24 bit colour mode providing over 16 million colours (select with COLOUR_24).

### 51.3.9 QL Colour Mode (COLOUR_QL)

This is similar to MODE 4 under the Standard QL Colour Drivers and is provided for compatability. However all 8 standard MODE 8 colours are actually available. See INK for a list of the standard MODE 8 colours.

### 51.3.10 PAL Colour Mode (COLOUR_PAL)

This allows programs to use 256 colours - it is the simplest mode to use, since a standard PAL Colour Value is used by any standard colour commands, such as INK, to describe all 256 colours on all implementations (including Aurora).

The table on the following pages describes all 256 colours with the PAL Colour Value and their Native Colour Value in decimal, hexadecimal and binary (see below).

You can use PALETTE_8 to change the 256 colours available.

### 51.3.11 Native Colour Mode (COLOUR_NATIVE)

As with the Q40/Q60, this allows programs to use 65536 colours. However, the Native Colour Values required for INK, STRIP, PAPER etc. depend upon the hardware (ie. they are different to Q40/Q60 values). The colour is described by the actual value which would be POKEd into the video memory, hence two bytes (a word) represent the colour of one pixel on the Monitor's screen. It is therefore easier to use Hexadecimal values to represent each colour.

The bits in the word represent the amount of GREEN, RED and BLUE to be used for each pixel, in the format **RRRRRGGG GGGBBBBB**, where:

- G is Green (6 bits)
- R is Red (5 bits)
- B is Blue (5 bits)

The table on the following pages describes the first 256 colours with the PAL Colour Value and their Native Colour Value in decimal, hexadecimal and binary (see below).

**NOTE:** When the values are entered direct into memory with a POKE command or machine code routine, due to the organisation of memory on a PC, it is necessary to enter the low byte before the high byte. As a result, the value for red, in binary 11111000 00000000 (INK $F800) is entered as POKE address,$00F8.

### 51.3.12 24 Bit Colour Mode (COLOUR_24)

This is supported only on PCs with 24 bit graphics cards. However, it is essential to understand this mode as commands such as PALETTE_8 and PALETTE_QL expect colours to be described in this format. The details appear later in this Appendix.

**Colour Table**

Due to the range of colours available, it is hard to describe each colour; therefore it will require experimentation to obtain the correct colours. The following table details the PAL Colour Value and Native Colour Value for each colour which need to be used for INK and similar commands.

**NOTE** This does not correspond with the orignal QL colour scheme!.

We have tried to list the most widely used ones (INK 0 to INK 63 under COLOUR_PAL) grouped into the different colours followed by the values for the remainder of the first 256 colours. Conversion of the values to binary gives a clue as to how this colour scheme works:

| PAL Colour Value | Colour Name | RRRRRGGG GGGBBBBB Native Value (Hex) |
|---|---|---|
| 0 | Black | $0000 |
| 1 | White | $FFFF |
| 8 | Dark Slate | $2124 |
| 9 | Slate Grey | $4A49 |
| 10 | Dark Grey | $6B6D |
| 11 | Grey | $9492 |
| 12 | Light Grey | $B5B6 |
| 13 | Ash Grey | $DEDB |
| 58 | Cerise | $9009 |
| 14 | Dark Red | $9000 |
| 2 | Red | $F800 |

Table 6 – continued from previous page

| PAL Colour Value | Colour Name | RRRRRGGG GGGBBBBB Native Value (Hex) |
|---|---|---|
| 63 | Deep Purple | $4812 |
| 51 | Plum | $692D |
| 20 | Purple | $9012 |
| 26 | Mauve | $901F |
| 57 | Faded Purple | $9256 |
| 52 | Dusky Pink | $B252 |
| 5 | Magenta | $F81F |
| 21 | Shocking Pink | $F812 |
| 45 | Dull Pink | $B376 |
| 32 | Pink | $FDBB |
| 31 | Rose Pink | $FDBF |
| 34 | Pastel Pink | $FEDB |
| 39 | Pastel Rose | $FEDF |
| 27 | Peach | $FDB6 |
| 50 | Midnight Blue | $212D |
| 19 | Dark Blue | $0012 |
| 4 | Blue | $001F |
| 62 | Ultramarine | $0252 |
| 49 | Dusky Blue | $236D |
| 44 | Steel Blue | $6B76 |
| 18 | Sea Blue | $0492 |
| 25 | Bright Blue | $049F |
| 56 | Dull Blue | $4C96 |
| 43 | Dull Cyan | $6DB6 |
| 7 | Cyan | $07FF |
| 29 | Light Blue | $B7FF |
| 30 | Sky Blue | $B5BF |
| 38 | Pastel Blue | $DEDF |
| 37 | Pastel Cyan | $DFFF |
| 48 | Dusky Green | $2364 |
| 60 | Grass Green | $4C80 |
| 17 | Dark Green | $0480 |
| 54 | Avocado | $95A9 |
| 61 | Sea Green | $0489 |
| 42 | Dull Green | $6DAD |
| 3 | Green | $07E0 |
| 23 | Lime Green | $97E0 |
| 24 | Apple Green | $07F2 |
| 55 | Dull Turquoise | $4DB2 |
| 41 | Light Khaki | $B5AD |
| 15 | Light Green | $B7F6 |
| 36 | Pastel Green | $DFFB |
| 46 | Brown | $6924 |
| 59 | Tan | $9240 |
| 6 | Yellow | $FFE0 |
| 22 | Orange | $FC80 |
| 16 | Mustard | $9480 |

Table 6 – continued from previous page

| PAL Colour Value | Colour Name | RRRRRGGG GGGBBBBB Native Value (Hex) |
|---|---|---|
| 47 | Khaki | $6B64 |
| 53 | Buff | $B489 |
| 40 | Brick | $B36D |
| 33 | Beige | $FED6 |
| 28 | Light Yellow | $FFF6 |
| 35 | Pastel Yellow | $FFFB |

The remainder of the first 256 colours are mapped as follows (we would welcome names for each of these colours):

| PAL Colour Value | RRRRRGGG GGGBBBBB Native Value (Hex) | Native Value (Binary) |
|---|---|---|
| 64 | $0009 | 00000000 00001001 |
| 65 | $2004 | 00100000 00000100 |
| 66 | $200D | 00100000 00001101 |
| 67 | $2016 | 00100000 00010110 |
| 68 | $201F | 00100000 00011111 |
| 69 | $4800 | 01001000 00000000 |
| 70 | $4809 | 01001000 00001001 |
| 71 | $481B | 01001000 00011011 |
| 72 | $6804 | 01101000 00000100 |
| 73 | $680D | 01101000 00001101 |
| 74 | $6816 | 01101000 00010110 |
| 75 | $681F | 01101000 00011111 |
| 76 | $B004 | 10110000 00000100 |
| 77 | $B00D | 10110000 00001101 |
| 78 | $B016 | 10110000 00010110 |
| 79 | $B01F | 10110000 00011111 |
| 80 | $D800 | 11011000 00000000 |
| 81 | $D809 | 11011000 00001001 |
| 82 | $D812 | 11011000 00010010 |
| 83 | $D81B | 11011000 00011011 |
| 84 | $F80D | 11111000 00001101 |
| 85 | $0120 | 00000001 00100000 |
| 86 | $0129 | 00000001 00101001 |
| 87 | $0132 | 00000001 00110010 |
| 88 | $013B | 00000001 00111011 |
| 89 | $2136 | 00100001 00110110 |
| 90 | $213F | 00100001 00111111 |
| 91 | $4920 | 01001001 00100000 |
| 92 | $4929 | 01001001 00101001 |
| 93 | $4932 | 01001001 00110010 |
| 94 | $493B | 01001001 00111011 |
| 95 | $6936 | 01101001 00110110 |
| 96 | $693F | 01101001 00111111 |

continues on next page

Table 7 – continued from previous page

| PAL Colour Value | RRRRRGGG GGGBBBBB<br>Native Value (Hex) | Native Value (Binary) |
|---|---|---|
| 97 | $9120 | 10010001 00100000 |
| 98 | $9129 | 10010001 00101001 |
| 99 | $9132 | 10010001 00110010 |
| 100 | $913B | 10010001 00111011 |
| 101 | $B124 | 10110001 00100100 |
| 102 | $B12D | 10110001 00101101 |
| 103 | $B136 | 10110001 00110110 |
| 104 | $B13F | 10110001 00111111 |
| 105 | $D920 | 11011001 00100000 |
| 106 | $D929 | 11011001 00101001 |
| 107 | $D932 | 11011001 00110010 |
| 108 | $D93B | 11011001 00111011 |
| 109 | $F924 | 11111001 00100100 |
| 110 | $F92D | 11111001 00101101 |
| 111 | $F936 | 11111001 00110110 |
| 112 | $F93F | 11111001 00111111 |
| 113 | $0240 | 00000010 01000000 |
| 114 | $0249 | 00000010 01001001 |
| 115 | $025B | 00000010 01011011 |
| 116 | $2244 | 00100010 01000100 |
| 117 | $224D | 00100010 01001101 |
| 118 | $2256 | 00100010 01010110 |
| 119 | $225F | 00100010 01011111 |
| 120 | $4A40 | 01001010 01000000 |
| 121 | $4A52 | 01001010 01010010 |
| 122 | $4A5B | 01001010 01011011 |
| 123 | $6A44 | 01101010 01000100 |
| 124 | $6A4D | 01101010 01001101 |
| 125 | $6456 | 01100100 01010110 |
| 126 | $6A5F | 01101010 01011111 |
| 127 | $9249 | 10010010 01001001 |
| 128 | $925B | 10010010 01011011 |
| 129 | $B244 | 10110010 01000100 |
| 130 | $B24D | 10110010 01001101 |
| 131 | $B25F | 10110010 01011111 |
| 132 | $DA40 | 11011010 01000000 |
| 133 | $DA49 | 11011010 01001001 |
| 134 | $DA52 | 11011010 01010010 |
| 135 | $DA5B | 11011010 01011011 |
| 136 | $FA44 | 11111010 01000100 |
| 137 | $FA4D | 11111010 01001101 |
| 138 | $FA56 | 11111010 01010110 |
| 139 | $FA5F | 11111010 01011111 |
| 140 | $0360 | 00000011 01100000 |
| 141 | $0369 | 00000011 01101001 |
| 142 | $0372 | 00000011 01110010 |
| 143 | $037B | 00000011 01111011 |

continues on next page

Table 7 – continued from previous page

| PAL Colour Value | RRRRRGGG GGGBBBBB<br>Native Value (Hex) | Native Value (Binary) |
|---|---|---|
| 144 | $2376 | 00100011 01110110 |
| 145 | $237F | 00100011 01111111 |
| 146 | $4B60 | 01001011 01100000 |
| 147 | $4B69 | 01001011 01101001 |
| 148 | $4B72 | 01001011 01110010 |
| 149 | $4B7B | 01001011 01111011 |
| 150 | $6B7F | 01101011 01111111 |
| 151 | $9360 | 10010011 01100000 |
| 152 | $9369 | 10010011 01101001 |
| 153 | $9372 | 10010011 01110010 |
| 154 | $937B | 10010011 01111011 |
| 155 | $B364 | 10110011 01100100 |
| 156 | $B37F | 10110011 01111111 |
| 157 | $DB60 | 11011011 01100000 |
| 158 | $DB69 | 11011011 01101001 |
| 159 | $DB72 | 11011011 01110010 |
| 160 | $DB7B | 11011011 01111011 |
| 161 | $FB64 | 11111011 01100100 |
| 162 | $FB6D | 11111011 01101101 |
| 163 | $FB76 | 11111011 01110110 |
| 164 | $FB7F | 11111011 01111111 |
| 165 | $2484 | 00100100 10000100 |
| 166 | $248D | 00100100 10001101 |
| 167 | $2496 | 00100100 10010110 |
| 168 | $249F | 00100100 10011111 |
| 169 | $4C89 | 01001100 10001001 |
| 170 | $4C9B | 01001100 10011011 |
| 171 | $6C84 | 01101100 10000100 |
| 172 | $6C8D | 01101100 10001101 |
| 173 | $6C96 | 01101100 10010110 |
| 174 | $6C9F | 01101100 10011111 |
| 175 | $9489 | 10010100 10001001 |
| 176 | $948B | 10010100 10001011 |
| 177 | $B484 | 10110100 10000100 |
| 178 | $B496 | 10110100 10010110 |
| 179 | $B49F | 10110100 10011111 |
| 180 | $DC80 | 11011100 10000000 |
| 181 | $DC89 | 11011100 10001001 |
| 182 | $DC92 | 11011100 10010010 |
| 183 | $DC9B | 11011100 10011011 |
| 184 | $FC8D | 11111100 10001101 |
| 185 | $FC96 | 11111100 10010110 |
| 186 | $FC9F | 11111100 10011111 |
| 187 | $05A0 | 00000101 10100000 |
| 188 | $05A9 | 00000101 10101001 |
| 189 | $05B2 | 00000101 10110010 |
| 190 | $05BB | 00000101 10111011 |

continues on next page

Table 7 – continued from previous page

| PAL Colour Value | RRRRRGGG GGGBBBBB<br>Native Value (Hex) | Native Value (Binary) |
|---|---|---|
| 191 | $25A4 | 00100101 10100100 |
| 192 | $25AD | 00100101 10101101 |
| 193 | $25B6 | 00100101 10110110 |
| 194 | $25BF | 00100101 10111111 |
| 195 | $4DA0 | 01001101 10100000 |
| 196 | $4DA9 | 01001101 10101001 |
| 197 | $4DBB | 01001101 10111011 |
| 198 | $6DA4 | 01101101 10100100 |
| 199 | $6DBF | 01101101 10111111 |
| 200 | $95A0 | 10010101 10100000 |
| 201 | $95B2 | 10010101 10110010 |
| 202 | $95BB | 10010101 10111011 |
| 203 | $B5A4 | 10110101 10100100 |
| 204 | $DDA0 | 11011101 10100000 |
| 205 | $DDA9 | 11011101 10101001 |
| 206 | $DDB2 | 11011101 10110010 |
| 207 | $DDBB | 11011101 10111011 |
| 208 | $FDA4 | 11111101 10100100 |
| 209 | $FDAD | 11111101 10101101 |
| 210 | $06C0 | 00000110 11000000 |
| 211 | $06C9 | 00000110 11001001 |
| 212 | $06D2 | 00000110 11010010 |
| 213 | $06DB | 00000110 11011011 |
| 214 | $26C4 | 00100110 11000100 |
| 215 | $26CD | 00100110 11001101 |
| 216 | $26D6 | 00100110 11010110 |
| 217 | $26DF | 00100110 11011111 |
| 218 | $4EC0 | 01001110 11000000 |
| 219 | $4EC9 | 01001110 11001001 |
| 220 | $4ED2 | 01001110 11010010 |
| 221 | $4EDB | 01001110 11011011 |
| 222 | $6EC4 | 01101110 11000100 |
| 223 | $6ECD | 01101110 11001101 |
| 224 | $6ED6 | 01101110 11010110 |
| 225 | $6EDF | 01101110 11011111 |
| 226 | $96C0 | 10010110 11000000 |
| 227 | $96C9 | 10010110 11001001 |
| 228 | $96D2 | 10010110 11010010 |
| 229 | $96DB | 10010110 11011011 |
| 230 | $B6C4 | 10110110 11000100 |
| 231 | $B6CD | 10110110 11001101 |
| 232 | $B6D6 | 10110110 11010110 |
| 233 | $B6DF | 10110110 11011111 |
| 234 | $DEC9 | 11011110 11001001 |
| 235 | $DED2 | 11011110 11010010 |
| 236 | $FECD | 11111110 11001101 |
| 237 | $07E9 | 00000111 11101001 |

continues on next page

Table 7 – continued from previous page

| PAL Colour Value | RRRRRGGG GGGBBBBB<br>Native Value (Hex) | Native Value (Binary) |
|---|---|---|
| 238 | $27ED | 00100111 11101101 |
| 239 | $27F6 | 00100111 11110110 |
| 240 | $4FE0 | 01001111 11100000 |
| 241 | $4FE9 | 01001111 11101001 |
| 242 | $4FF2 | 01001111 11110010 |
| 243 | $4FFB | 01001111 11111011 |
| 244 | $6FE4 | 01101111 11100100 |
| 245 | $6FED | 01101111 11101101 |
| 246 | $6FF6 | 01101111 11110110 |
| 247 | $6FFF | 01101111 11111111 |
| 248 | $97E9 | 10010111 11101001 |
| 249 | $97F2 | 10010111 11110010 |
| 250 | $97FB | 10010111 11111011 |
| 251 | $B7E4 | 10110111 11100100 |
| 252 | $B7ED | 10110111 11101101 |
| 253 | $DFE9 | 11011111 11101001 |
| 254 | $DFF2 | 11011111 11110010 |
| 255 | $FFED | 11111111 11101101 |

### 51.3.13 Q40/Q60 Enhanced Colour Drivers

SMSQ/E v2.98+ provides various colour modes for the Q40 and Q60 computers. You can configure SMSQ/E to start with either the Standard QL Colour Drivers or the Enhanced Colour Drivers. If the Enhanced Colour Drivers are loaded, RMODE will return 33.

The Enhanced Colour Drivers support a QL 8 colour mode (selected with COLOUR_QL), a PAL Colour Mode providing 256 colours (selected with COLOUR_PAL) and a Native Colour Mode providing 65536 colours (select with COLOUR_NATIVE). As with the other implementations, 24 bit colours are used by commands such as PALETTE_8, although there is no 24 bit colour mode due to the limitations of the hardware.

### 51.3.14 QL Colour Mode (COLOUR_QL)

This is similar to MODE 4 under the Standard QL Colour Drivers and is provided for compatibility. However all 8 standard MODE 8 colours are actually available. See INK for a list of the standard MODE 8 colours.

### 51.3.15 PAL Colour Mode (COLOUR_PAL)

This allows programs to use 256 colours - it is the simplest mode to use, since a standard PAL Colour Value is used by any standard colour commands, such as INK, to describe all 256 colours on all implementations (including Aurora).

The table on the following pages describes all 256 colours with the PAL Colour Value and their Native Colour Value in decimal, hexadecimal and binary (see below).

You can use PALETTE_8 to change the 256 colours available.

### 51.3.16 Native Colour Mode (COLOUR_NATIVE)

As with QXL and QPC2, this allows programs to use 65536 colours. However, the Native Colour Values required for INK, STRIP, PAPER etc. depend upon the hardware (ie. they are different to the QPC2/QXL values). The colour is described by the actual value which would be POKEd into the video memory, hence two bytes (a word) represent the colour of one pixel on the Monitor's screen. It is therefore easier to use Hexadecimal values to represent each colour.

The bits in the word represent the amount of GREEN, RED and BLUE to be used for each pixel, in the format **GGGGGRRR RRBBBBBW**, where:

- G is Green (5 bits)
- R Red (5 bits)
- B Blue (5 bits)
- W White

The table on the following pages describes the first 256 colours with the PAL Colour Value and their Native Colour Value in decimal, hexadecimal and binary (see below).

**Colour Table**

Due to the range of colours available, it is hard to describe each colour; therefore it will require experimentation to obtain the correct colours. The following table details the PAL Colour Value and Native Colour Value for each colour which need to be used for INK and similar commands (**NOTE** this does not correspond with the original QL colour scheme!). We have tried to list the most widely used ones (INK 0 to INK 63 under COLOUR_PAL) grouped into the different colours followed by the values for the remainder of the first 256 colours. Conversion of the values to binary gives a clue as to how this colour scheme works:

| PAL Colour Value | GGGGGRRR RRBBBBBW Colour Name | Native Value Hex | Native Value Binary |
|---|---|---|---|
| 0 | Black | $0000 | 00000000 00000000 |
| 1 | White | $FFFF | 11111111 11111111 |
| 8 | Dark Slate | $2108 | 00100001 00001000 |
| 9 | Slate Grey | $4A53 | 01001010 01010011 |
| 10 | Dark Grey | $6B5B | 01101011 01011011 |
| 11 | Grey | $94A4 | 10010100 10100100 |
| 12 | Light Grey | $B5AC | 10110101 10101100 |
| 13 | Ash Grey | $DEF7 | 11011110 11110111 |
| 58 | Cerise | $0492 | 00000100 10010010 |
| 14 | Dark Red | $0480 | 00000100 10000000 |
| 2 | Red | $07C0 | 00000111 11000000 |
| 63 | Deep Purple | $0264 | 00000010 01100100 |
| 51 | Plum | $235B | 00100011 01011011 |
| 20 | Purple | $04A4 | 00000100 10100100 |
| 26 | Mauve | $04BE | 00000100 10111110 |
| 57 | Faded Purple | $4CAC | 01001100 10101100 |
| 52 | Dusky Pink | $4DA4 | 01001101 10100100 |
| 5 | Magenta | $07FF | 00000111 11111111 |
| 21 | Shocking Pink | $07E4 | 00000111 11100100 |
| 45 | Dull Pink | $6DAC | 01101101 10101100 |

continues on next page

Table 8 – continued from previous page

| PAL Colour Value | GGGGGRRR RRBBBBBW Colour Name | Native Value Hex | Native Value Binary |
|---|---|---|---|
| 32 | Pink | $B7F7 | 10110111 11110111 |
| 31 | Rose Pink | $B7FF | 10110111 11111111 |
| 34 | Pastel Pink | $DFF7 | 11011111 11110111 |
| 39 | Pastel Rose | $DFFF | 11011111 11111111 |
| 27 | Peach | $B7EC | 10110111 11101100 |
| 50 | Midnight Blue | $211A | 00100001 00011010 |
| 19 | Dark Blue | $0024 | 00000000 00100100 |
| 4 | Blue | $003E | 00000000 00111110 |
| 62 | Ultramarine | $4824 | 01001000 00100100 |
| 49 | Dusky Blue | $691B | 01101001 00011011 |
| 44 | Steel Blue | $6B6D | 01101011 01101101 |
| 18 | Sea Blue | $9024 | 10010000 00100100 |
| 25 | Bright Blue | $903E | 10010000 00111110 |
| 56 | Dull Blue | $926C | 10010010 01101100 |
| 43 | Dull Cyan | $B36C | 10110011 01101100 |
| 7 | Cyan | $F83F | 11111000 00111111 |
| 29 | Light Blue | $FDBF | 11111101 10111111 |
| 30 | Sky Blue | $B5BE | 10110101 10111110 |
| 38 | Pastel Blue | $DEFF | 11011110 11111111 |
| 37 | Pastel Cyan | $FEFF | 11111110 11111111 |
| 48 | Dusky Green | $6908 | 01101001 00001000 |
| 60 | Grass Green | $9240 | 10010010 01000000 |
| 17 | Dark Green | $9000 | 10010000 00000000 |
| 54 | Avocado | $B492 | 10110100 10010010 |
| 61 | Sea Green | $9012 | 10010000 00010010 |
| 42 | Dull Green | $B35B | 10110011 01011011 |
| 3 | Green | $F800 | 11111000 00000000 |
| 23 | Lime Green | $FC80 | 11111100 10000000 |
| 24 | Apple Green | $F824 | 11111000 00100100 |
| 55 | Dull Turquoise | $B264 | 10110010 01100100 |
| 41 | Light Khaki | $B59A | 10110101 10011010 |
| 15 | Light Green | $FDAC | 11111101 10101100 |
| 36 | Pastel Green | $FEF7 | 11111110 11110111 |
| 46 | Brown | $2348 | 00100011 01001000 |
| 59 | Tan | $4C80 | 01001100 10000000 |
| 6 | Yellow | $FFC1 | 11111111 11000001 |
| 22 | Orange | $97C0 | 10010111 11000000 |
| 16 | Mustard | $9480 | 10010100 10000000 |
| 47 | Khaki | $6B49 | 01101011 01001001 |
| 53 | Buff | $9592 | 10010101 10010010 |
| 40 | Brick | $6D9B | 01101101 10011011 |
| 33 | Beige | $DFED | 11011111 11101101 |
| 28 | Light Yellow | $FFED | 11111111 11101101 |
| 35 | Pastel Yellow | $FFF7 | 11111111 11110111 |

The remainder of the first 256 colours are mapped as follows (we would welcome names for each of these colours):

| PAL Colour Value | GGGGGRRR RRBBBBBW<br>Native Value Hex | Native Value Binary |
|---|---|---|
| 64 | $0012 | 00000000 00010010 |
| 65 | $0108 | 00000001 00001000 |
| 66 | $011A | 00000001 00011010 |
| 67 | $012C | 00000001 00101100 |
| 68 | $013E | 00000001 00111110 |
| 69 | $0240 | 00000010 01000000 |
| 70 | $0253 | 00000010 01010011 |
| 71 | $0277 | 00000010 01110111 |
| 72 | $0348 | 00000011 01001000 |
| 73 | $035B | 00000011 01011011 |
| 74 | $036C | 00000011 01101100 |
| 75 | $037F | 00000011 01111111 |
| 76 | $0588 | 00000101 10001000 |
| 77 | $059A | 00000101 10011010 |
| 78 | $05AC | 00000101 10101100 |
| 79 | $05BE | 00000101 10111110 |
| 80 | $06C0 | 00000110 11000000 |
| 81 | $06D3 | 00000110 11010011 |
| 82 | $06E4 | 00000110 11100100 |
| 83 | $06F7 | 00000110 11110111 |
| 84 | $07DB | 00000111 11011011 |
| 85 | $2000 | 00100000 00000000 |
| 86 | $2012 | 00100000 00010010 |
| 87 | $2024 | 00100000 00100100 |
| 88 | $2036 | 00100000 00110110 |
| 89 | $212C | 00100001 00101100 |
| 90 | $213E | 00100001 00111110 |
| 91 | $2240 | 00100010 01000000 |
| 92 | $2253 | 00100010 01010011 |
| 93 | $2264 | 00100010 01100100 |
| 94 | $2277 | 00100010 01110111 |
| 95 | $236C | 00100011 01101100 |
| 96 | $237F | 00100011 01111111 |
| 97 | $2480 | 00100100 10000000 |
| 98 | $2492 | 00100100 10010010 |
| 99 | $24A4 | 00100100 10100100 |
| 100 | $24B6 | 00100100 10110110 |
| 101 | $2588 | 00100101 10001000 |
| 102 | $259A | 00100101 10011010 |
| 103 | $25AC | 00100101 10101100 |
| 104 | $25BE | 00100101 10111110 |
| 105 | $26C0 | 00100110 11000000 |
| 106 | $26D3 | 00100110 11010011 |
| 107 | $26E4 | 00100110 11100100 |
| 108 | $26F7 | 00100110 11110111 |
| 109 | $27C8 | 00100111 11001000 |
| 110 | $27DB | 00100111 11011011 |

continues on next page

Table 9 – continued from previous page

| PAL Colour Value | GGGGGRRR RRBBBBBW<br>Native Value Hex | Native Value Binary |
|---|---|---|
| 111 | $27EC | 00100111 11101100 |
| 112 | $27FF | 00100111 11111111 |
| 113 | $4800 | 01001000 00000000 |
| 114 | $4813 | 01001000 00010011 |
| 115 | $4837 | 01001000 00110111 |
| 116 | $4908 | 01001001 00001000 |
| 117 | $491B | 01001001 00011011 |
| 118 | $492C | 01001001 00101100 |
| 119 | $493F | 01001001 00111111 |
| 120 | $4A41 | 01001010 01000001 |
| 121 | $4A65 | 01001010 01100101 |
| 122 | $4A77 | 01001010 01110111 |
| 123 | $4B49 | 01001011 01001001 |
| 124 | $4B5B | 01001011 01011011 |
| 125 | $8B2C | 10001011 00101100 |
| 126 | $4B7F | 01001011 01111111 |
| 127 | $4C93 | 01001100 10010011 |
| 128 | $4CB7 | 01001100 10110111 |
| 129 | $4D88 | 01001101 10001000 |
| 130 | $4D9B | 01001101 10011011 |
| 131 | $4DBF | 01001101 10111111 |
| 132 | $4EC1 | 01001110 11000001 |
| 133 | $4ED3 | 01001110 11010011 |
| 134 | $4EE5 | 01001110 11100101 |
| 135 | $4EF7 | 01001110 11110111 |
| 136 | $4FC9 | 01001111 11001001 |
| 137 | $4FDB | 01001111 11011011 |
| 138 | $4FED | 01001111 11101101 |
| 139 | $4FFF | 01001111 11111111 |
| 140 | $6800 | 01101000 00000000 |
| 141 | $6813 | 01101000 00010011 |
| 142 | $6824 | 01101000 00100100 |
| 143 | $6837 | 01101000 00110111 |
| 144 | $692C | 01101001 00101100 |
| 145 | $693F | 01101001 00111111 |
| 146 | $6A41 | 01101010 01000001 |
| 147 | $6A53 | 01101010 01010011 |
| 148 | $6A65 | 01101010 01100101 |
| 149 | $6A77 | 01101010 01110111 |
| 150 | $6B7F | 01101011 01111111 |
| 151 | $6C80 | 01101100 10000000 |
| 152 | $6C93 | 01101100 10010011 |
| 153 | $6CA4 | 01101100 10100100 |
| 154 | $6CB7 | 01101100 10110111 |
| 155 | $6D88 | 01101101 10001000 |
| 156 | $6DBF | 01101101 10111111 |
| 157 | $6EC1 | 01101110 11000001 |

continues on next page

Table 9 – continued from previous page

| PAL Colour Value | GGGGGRRR RRBBBBBW<br>Native Value Hex | Native Value Binary |
|---|---|---|
| 158 | $6ED3 | 01101110 11010011 |
| 159 | $6EE5 | 01101110 11100101 |
| 160 | $6EF7 | 01101110 11110111 |
| 161 | $6FC9 | 01101111 11001001 |
| 162 | $6FDB | 01101111 11011011 |
| 163 | $6FED | 01101111 11101101 |
| 164 | $6FFF | 01101111 11111111 |
| 165 | $9108 | 10010001 00001000 |
| 166 | $911A | 10010001 00011010 |
| 167 | $912C | 10010001 00101100 |
| 168 | $913E | 10010001 00111110 |
| 169 | $9253 | 10010010 01010011 |
| 170 | $9277 | 10010010 01110111 |
| 171 | $9348 | 10010011 01001000 |
| 172 | $935B | 10010011 01011011 |
| 173 | $936C | 10010011 01101100 |
| 174 | $937F | 10010011 01111111 |
| 175 | $9492 | 10010100 10010010 |
| 176 | $9496 | 10010100 10010110 |
| 177 | $9588 | 10010101 10001000 |
| 178 | $95AC | 10010101 10101100 |
| 179 | $95BE | 10010101 10111110 |
| 180 | $96C0 | 10010110 11000000 |
| 181 | $96D3 | 10010110 11010011 |
| 182 | $96E4 | 10010110 11100100 |
| 183 | $96F7 | 10010110 11110111 |
| 184 | $97DB | 10010111 11011011 |
| 185 | $97EC | 10010111 11101100 |
| 186 | $97FF | 10010111 11111111 |
| 187 | $B000 | 10110000 00000000 |
| 188 | $B012 | 10110000 00010010 |
| 189 | $B024 | 10110000 00100100 |
| 190 | $B036 | 10110000 00110110 |
| 191 | $B108 | 10110001 00001000 |
| 192 | $B11A | 10110001 00011010 |
| 193 | $B12C | 10110001 00101100 |
| 194 | $B13E | 10110001 00111110 |
| 195 | $B240 | 10110010 01000000 |
| 196 | $B253 | 10110010 01010011 |
| 197 | $B277 | 10110010 01110111 |
| 198 | $B348 | 10110011 01001000 |
| 199 | $B37F | 10110011 01111111 |
| 200 | $B480 | 10110100 10000000 |
| 201 | $B4A4 | 10110100 10100100 |
| 202 | $B4B6 | 10110100 10110110 |
| 203 | $B588 | 10110101 10001000 |
| 204 | $B6C0 | 10110110 11000000 |

continues on next page

Table 9 – continued from previous page

| PAL Colour Value | GGGGGRRR RRBBBBBW<br>Native Value Hex | Native Value Binary |
|---|---|---|
| 205 | $B6D3 | 10110110 11010011 |
| 206 | $B6E4 | 10110110 11100100 |
| 207 | $B6F7 | 10110110 11110111 |
| 208 | $B7C8 | 10110111 11001000 |
| 209 | $B7DB | 10110111 11011011 |
| 210 | $D800 | 11011000 00000000 |
| 211 | $D813 | 11011000 00010011 |
| 212 | $D824 | 11011000 00100100 |
| 213 | $D837 | 11011000 00110111 |
| 214 | $D908 | 11011001 00001000 |
| 215 | $D91B | 11011001 00011011 |
| 216 | $D92C | 11011001 00101100 |
| 217 | $D93F | 11011001 00111111 |
| 218 | $DA41 | 11011010 01000001 |
| 219 | $DA53 | 11011010 01010011 |
| 220 | $DA65 | 11011010 01100101 |
| 221 | $DA77 | 11011010 01110111 |
| 222 | $DB49 | 11011011 01001001 |
| 223 | $DB5B | 11011011 01011011 |
| 224 | $DB6D | 11011011 01101101 |
| 225 | $DB7F | 11011011 01111111 |
| 226 | $DC80 | 11011100 10000000 |
| 227 | $DC93 | 11011100 10010011 |
| 228 | $DCA4 | 11011100 10100100 |
| 229 | $DCB7 | 11011100 10110111 |
| 230 | $DD88 | 11011101 10001000 |
| 231 | $DD9B | 11011101 10011011 |
| 232 | $DDAC | 11011101 10101100 |
| 233 | $DDBF | 11011101 10111111 |
| 234 | $DED3 | 11011110 11010011 |
| 235 | $DEE5 | 11011110 11100101 |
| 236 | $DFDB | 11011111 11011011 |
| 237 | $F813 | 11111000 00010011 |
| 238 | $F91B | 11111001 00011011 |
| 239 | $F92C | 11111001 00101100 |
| 240 | $FA41 | 11111010 01000001 |
| 241 | $FA53 | 11111010 01010011 |
| 242 | $FA65 | 11111010 01100101 |
| 243 | $FA77 | 11111010 01110111 |
| 244 | $FB49 | 11111011 01001001 |
| 245 | $FB5B | 11111011 01011011 |
| 246 | $FB6D | 11111011 01101101 |
| 247 | $FB7F | 11111011 01111111 |
| 248 | $FC93 | 11111100 10010011 |
| 249 | $FCA4 | 11111100 10100100 |
| 250 | $FCB7 | 11111100 10110111 |
| 251 | $FD88 | 11111101 10001000 |

continues on next page

Table 9 – continued from previous page

| PAL Colour Value | GGGGGRRR RRBBBBBW Native Value Hex | Native Value Binary |
|---|---|---|
| 252 | $FD9B | 11111101 10011011 |
| 253 | $FED3 | 11111110 11010011 |
| 254 | $FEE5 | 11111110 11100101 |
| 255 | $FFDB | 11111111 11011011 |

### 51.3.17 24 Bit Enhanced Colour Drivers

Although only available as a Colour Mode on QPC2 and the QXL, this true colour (24 bit) mode is used by commands such as PALETTE_QL and PALETTE_8 to describe approx 16 million colours in detail.

Here, every four bytes (a longword) represent one pixel on the Monitor's screen.

The bits are combined to represent the amount of GREEN, RED and BLUE to be used for each pixel, in the following format **rrrrrrrr gggggggg bbbbbbbb xxxxxxxx**, where:

- G is Green (8 bits)
- R is Red (8 bits)
- B is Blue (8 bits)
- X is Unused

In the table below, the colours represented by each of the first 256 PAL colours (0 to 255) closely resembles those generated under the 256 Colour Mode on the Aurora, however, due to the way in which colour is stored, it is necessary to look up the comparative Hexadecimal value for each colour which would need to be POKEd into memory.

You cannot use the PAL colour number as a parameter for INK (and other commands) due to the fact that this is limited to 256 - use the hexadecimal 24 bit value instead.

Again, the colours are hard to describe due to the range and therefore require experimentation to obtain the correct colours. However, the following table details the corresponding INK parameter to use to achieve that colour (*NOTE* this does not correspond with the original QL colour scheme!). It is not possible to list all of the 16 million colours, therefore we have tried to list the most widely used ones (INK 0 to INK 63) grouped into the different colours and the values for the rest of the colours in the range INK 64 to INK 255.

| PAL Colour Value | Colour Name | 24 bit Value (Hex) |
|---|---|---|
| 0 | Black | $000000 |
| 1 | White | $FFFFFF |
| 8 | Dark Slate | $242424 |
| 9 | Slate Grey | $494949 |
| 10 | Dark Grey | $6D6D6D |
| 11 | Grey | $929292 |
| 12 | Light Grey | $B6B6B6 |
| 13 | Ash Grey | $DBDBDB |
| 58 | Cerise | $920049 |
| 14 | Dark Red | $920000 |
| 2 | Red | $FF0000 |

continues on next page

Table 10 – continued from previous page

| PAL Colour Value | Colour Name | 24 bit Value (Hex) |
|---|---|---|
| 63 | Deep Purple | $490092 |
| 51 | Plum | $6D246D |
| 20 | Purple | $920092 |
| 26 | Mauve | $9200FF |
| 57 | Faded Purple | $9249B6 |
| 52 | Dusky Pink | $B64992 |
| 5 | Magenta | $FF00FF |
| 21 | Shocking Pink | $FF0092 |
| 45 | Dull Pink | $B66DB6 |
| 32 | Pink | $FFB6DB |
| 31 | Rose Pink | $FFB6FF |
| 34 | Pastel Pink | $FFDBDB |
| 39 | Pastel Rose | $FFDBFF |
| 27 | Peach | $FFB6B6 |
| 50 | Midnight Blue | $24246D |
| 19 | Dark Blue | $000092 |
| 4 | Blue | $0000FF |
| 62 | Ultramarine | $004992 |
| 49 | Dusky Blue | $246D6D |
| 44 | Steel Blue | $6D6DB6 |
| 18 | Sea Blue | $009292 |
| 25 | Bright Blue | $0092FF |
| 56 | Dull Blue | $4992B6 |
| 43 | Dull Cyan | $6DB6B6 |
| 7 | Cyan | $00FFFF |
| 29 | Light Blue | $B6FFFF |
| 30 | Sky Blue | $B6B6FF |
| 38 | Pastel Blue | $DBDBFF |
| 37 | Pastel Cyan | $DBFFFF |
| 48 | Dusky Green | $246D24 |
| 60 | Grass Green | $499200 |
| 17 | Dark Green | $009200 |
| 54 | Avocado | $92B649 |
| 61 | Sea Green | $009249 |
| 42 | Dull Green | $6DB66D |
| 3 | Green | $00FF00 |
| 23 | Lime Green | $92FF00 |
| 24 | Apple Green | $00FF92 |
| 55 | Dull Turquoise | $49B692 |
| 41 | Light Khaki | $B6B66D |
| 15 | Light Green | $B6FFB6 |
| 36 | Pastel Green | $DBFFDB |
| 46 | Brown | $6D2424 |
| 59 | Tan | $924900 |
| 6 | Yellow | $FFFF00 |
| 22 | Orange | $FF9200 |
| 16 | Mustard | $929200 |
| 47 | Khaki | $6D6D24 |

continues on next page

Table 10 – continued from previous page

| PAL Colour Value | Colour Name | 24 bit Value (Hex) |
|---|---|---|
| 53 | Buff | $B69249 |
| 40 | Brick | $B66D6D |
| 33 | Beige | $FFDBB6 |
| 28 | Light Yellow | $FFFFB6 |
| 35 | Pastel Yellow | $FFFFDB |

The remainder of the first 256 colours are mapped as follows (we would welcome names for each of these colours):

| PAL Colour Value | 24 bit Value (Hex) |
|---|---|
| 64 | $000049 |
| 65 | $240024 |
| 66 | $24006D |
| 67 | $2400B6 |
| 68 | $2400FF |
| 69 | $490000 |
| 70 | $490049 |
| 71 | $4900DB |
| 72 | $6D0024 |
| 73 | $6D006D |
| 74 | $6D00B6 |
| 75 | $6D00FF |
| 76 | $B60024 |
| 77 | $B6006D |
| 78 | $B600B6 |
| 79 | $B600FF |
| 80 | $DB0000 |
| 81 | $DB0049 |
| 82 | $DB0092 |
| 83 | $DB00DB |
| 84 | $FF006D |
| 85 | $002400 |
| 86 | $002449 |
| 87 | $002492 |
| 88 | $0024DB |
| 89 | $2424B6 |
| 90 | $2424FF |
| 91 | $492400 |
| 92 | $492449 |
| 93 | $492492 |
| 94 | $4924DB |
| 95 | $6D24B6 |
| 96 | $6D24FF |
| 97 | $922400 |
| 98 | $922449 |
| 99 | $922492 |

continues on next page

Table 11 – continued from previous page

| PAL Colour Value | 24 bit Value (Hex) |
|---|---|
| 100 | $9224DB |
| 101 | $B62424 |
| 102 | $B6246D |
| 103 | $B624B6 |
| 104 | $B624FF |
| 105 | $DB2400 |
| 106 | $DB2449 |
| 107 | $DB2492 |
| 108 | $DB246D |
| 109 | $FF2424 |
| 110 | $FF246D |
| 111 | $FF24B6 |
| 112 | $FF24FF |
| 113 | $004900 |
| 114 | $004949 |
| 115 | $0049DB |
| 116 | $244924 |
| 117 | $24496D |
| 118 | $2449B6 |
| 119 | $2449FF |
| 120 | $494900 |
| 121 | $494992 |
| 122 | $4949DB |
| 123 | $6D4924 |
| 124 | $6D496D |
| 125 | $6D49B6 |
| 126 | $6D49FF |
| 127 | $924949 |
| 128 | $9249DB |
| 129 | $B64924 |
| 130 | $B6496D |
| 131 | $B649FF |
| 132 | $DB4900 |
| 133 | $DB4949 |
| 134 | $DB4992 |
| 135 | $DB49DB |
| 136 | $FF4924 |
| 137 | $FF496D |
| 138 | $FF49B6 |
| 139 | $FF49FF |
| 140 | $006D00 |
| 141 | $006D49 |
| 142 | $006D92 |
| 143 | $006DDB |
| 144 | $246DB6 |
| 145 | $246DFF |
| 146 | $496D00 |
| 147 | $496D49 |

continues on next page

Table 11 – continued from previous page

| PAL Colour Value | 24 bit Value (Hex) |
|---|---|
| 148 | $496D92 |
| 149 | $496DDB |
| 150 | $6D6DFF |
| 151 | $926D00 |
| 152 | $926D49 |
| 153 | $926D92 |
| 154 | $926DDB |
| 155 | $B66D24 |
| 156 | $B66DFF |
| 157 | $DB6D00 |
| 158 | $DB6D49 |
| 159 | $DB6D92 |
| 160 | $DB6DDB |
| 161 | $FF6D24 |
| 162 | $FF6D6D |
| 163 | $FF6DB6 |
| 164 | $FF6DFF |
| 165 | $249224 |
| 166 | $24926D |
| 167 | $2492B6 |
| 168 | $2492FF |
| 169 | $499249 |
| 170 | $4992DB |
| 171 | $6D9224 |
| 172 | $6D926D |
| 173 | $6D92B6 |
| 174 | $6D92FF |
| 175 | $929249 |
| 176 | $9292DB |
| 177 | $B69224 |
| 178 | $B692B6 |
| 179 | $B692FF |
| 180 | $DB9200 |
| 181 | $DB9249 |
| 182 | $DB9292 |
| 183 | $DB92DB |
| 184 | $FF926D |
| 185 | $FF92B6 |
| 186 | $FF92FF |
| 187 | $00B600 |
| 188 | $00B649 |
| 189 | $00B692 |
| 190 | $00B6DB |
| 191 | $24B624 |
| 192 | $24B66D |
| 193 | $24B6B6 |
| 194 | $24B6FF |
| 195 | $49B600 |

continues on next page

Table 11 – continued from previous page

| PAL Colour Value | 24 bit Value (Hex) |
|---|---|
| 196 | $49B649 |
| 197 | $49B6DB |
| 198 | $6DB624 |
| 199 | $6DB6FF |
| 200 | $92B600 |
| 201 | $92B692 |
| 202 | $92B6DB |
| 203 | $B6B624 |
| 204 | $DBB600 |
| 205 | $DBB649 |
| 206 | $DBB692 |
| 207 | $DBB6DB |
| 208 | $FFB624 |
| 209 | $FFB66D |
| 210 | $00DB00 |
| 211 | $00DB49 |
| 212 | $00DB92 |
| 213 | $00DBDB |
| 214 | $24DB24 |
| 215 | $24DB6D |
| 216 | $24DBB6 |
| 217 | $24DBFF |
| 218 | $49DB00 |
| 219 | $49DB49 |
| 220 | $49DB92 |
| 221 | $49DBDB |
| 222 | $6DDB24 |
| 223 | $6DDB6D |
| 224 | $6DDBB6 |
| 225 | $6DDBFF |
| 226 | $92DB00 |
| 227 | $92DB49 |
| 228 | $92DB92 |
| 229 | $92DBDB |
| 230 | $B6DB24 |
| 231 | $B6DB6D |
| 232 | $B6DBN6 |
| 233 | $B6DBFF |
| 234 | $DBDB49 |
| 235 | $DBDB92 |
| 236 | $FFDB6D |
| 237 | $00FF49 |
| 238 | $24FF6D |
| 239 | $24FFB6 |
| 240 | $49FF00 |
| 241 | $49FF49 |
| 242 | $49FF92 |
| 243 | $49FFDB |

continues on next page

Table 11 – continued from previous page

| PAL Colour Value | 24 bit Value (Hex) |
|---|---|
| 244 | $6DFF24 |
| 245 | $6DFF6D |
| 246 | $6DFFB6 |
| 247 | $6DFFFF |
| 248 | $92FF49 |
| 249 | $92FF92 |
| 250 | $92FFDB |
| 251 | $B6FF24 |
| 252 | $B6FF6D |
| 253 | $DBFF49 |
| 254 | $DBFF92 |
| 255 | $FFFF6D |

### 51.3.18 MISCELLANEOUS COLOUR MODES

#### MODE 2 (Monochrome Drivers)

Each byte represents eight pixels on the Monitor's screen calculated by looking at the status of each of the eight binary bits which make up a byte - if a bit is 1 (ON) then the corresponding pixel will be white, otherwise it will be black.

For example, if PEEK (SCR_BASE) returns the value 85, in binary this is represented by:

0 1 0 1 0 1 0 1

Therefore the top left of the Monitor screen will be showing eight alternating pixels of black and white.

**Example**

The following program will fill the screen with black and white vertical stripes:

```
100 MODE 2
110 FOR x=0 TO SCR_YLIM-1
120   FOR y=0 TO (SCR_XLIM-1)/4
130     POKE SCR_BASE+ (x*SCR_LLEN) + y,85
140   END FOR y
150 END FOR x
```

**SMSQ/E NOTE**

Under SMSQ/E v2.98+, Enhanced Colour Drivers, this becomes MODE 0.

#### MODE 12 (16 Colour Mode)

This is supported only by the THOR XVI and is similar to Aurora's 16 Colour Mode under the Enhanced Colour Drivers in that it replaces the Flash bit of MODE 8 by an Intensity bit which allows you to display 16 colours on screen at a time at a resolution of 256 x 256 pixels.

(We have no details of how this was implemented).

## 51.4 A16.4 USING HIGH RESOLUTION DISPLAYS

There is not much adaptation required in order to use QL screen resolutions in excess of 512x256 pixels - all of the normal commands work as you would expect. OUTLN includes an example of how to allow programs to re-size themselves up to the maximum resolution. However, there are some rules which you need to observe..

First of all, refer to Appendix 6.7 (Using High Resolution Screens) about some compatibility issues.

If a user chooses to run a program on a high resolution screen, they will find that the program will only occupy a small area of the screen. Even if the program allows you to resize it to take advantage of the larger screen, this may not be very satisfactory. Although graphics commands which work by reference to the graphics co-ordinates system (such as LINE) will take advantage of the larger windows provided by the program, thus giving the effect of enlarging any on-screen graphics, other commands which work by reference to the pixel co-ordinates system (such as BLOCK) will rely on the program to resize them specifically for the new screen. You may therefore find that the display of some programs is corrupted, unless the programmer has taken sufficient care.

One of the main problems is that text can still only be PRINTed on screen in the standard QL sizes, and therefore, any text PRINTed in CSIZE 0,0 will be very hard to read on a 800x600 screen (let alone a 1600x1200 screen), even with a 17" Monitor. Programs will therefore need to take this into account, possibly using the resize procedure to alter the character sizes used for text, or using the ProWeSS system from PROGS, which uses scalable fonts for output.

Refer to the SCALE command for a means of working out a relationship between graphics co-ordinates and pixel co- ordinates.

CHAPTER

FIFTYTWO

# A17 NETWORKS

A Network is a means of communicating with several other computers and sharing their resources. The way in which networks work on a QL or derivative depends upon the hardware being used.

Standard QLs and the THOR family of computers have two network ports which work in exactly the same way, and enable up to 64 computers to be connected together serially.

Unfortunately, the protocol used for the Network is peculiar to Sinclair machines and only QLs (including AURORA), QXLs, THORs and ZX Spectrums may be connected together in this way. This is known as QNet. If you only need to network these machines together then you can take advantage of several different options to get the most out of the original QNet.

To enable ATARIs and PCs to be connected together in a Network with Sinclair QLs, you will need to use the SERNET and MIDINET drivers which allow the machines to be connected via serial ports and MIDI ports respectively (although MIDI ports can only be connected between ATARI computers).

These networks will work alongside the standard QNet and therefore you can have several circles of computers networked together, all controlled by one central machine, say with some linked via SERNET, some via MIDINET and others linked via QNet.

Another option available to users wishing to join machines together in a Network is to use Amadeus Interlink (© Di - Ren), which allows you to connect up to 255 machines in a Network. These machines can be either QLs, THORs or PCs at present and this Network system is significantly faster than the original QNet.

Di-Ren also produce a QL-PC Fileserver which allows you to link a QL to a PC with a cable and then this allows the QL to use the PC's DOS devices (including floppy disks, CD-Rom drive, hard disks, Networks and Serial ports). This will allow you to use PC format disks and on later versions, you can even use the PC's screen to display the QL's output. This is therefore similar to a small Network device.

In DIY Toolkit Volume N there is a very interesting program NETPAL which allows one computer to completely control another computer - this works by machine 2 opening a channel on machine 1 where the user can enter his commands, which are then executed on machine 2. This can be very useful to allow you to use several QLs with only one Monitor (or keyboard), but unfortunately, this program does have its limitations due to problems with sending control characters over the Network (such as CTRL C) and also a CON device opened over the Network does not seem to delete characters from the screen properly..

## 52.1 A17.1 QNet

This system allows Standard QLs, QXLs, AURORAs, THOR computers and ZX Spectrums (with Interface 1 attached) to be connected together in a network. However, although data can be sent to a ZX Spectrum, the data is corrupted if the Spectrum tries to send more than one byte to another type of machine.

Minerva, SMSQ/E and Toolkit II all improve the reliability of this Network system. if you are not using Toolkit II, Minerva v1.96 is ideally required to ensure reliable input of data over the Network.

### 52.1.1 A17.1.1 Connecting Machines

To connect the machines, a cable must be used to connect one network port on each computer to one of the ports on the next computer in line. Having chained all of the computers together in this fashion, the Network should be completed by connecting the last computer in the chain to the first, thus completing a circle (although this is not always necessary).

The cable need only be twin core, and it is advisable in a large Network to use low-capacitance cable (eg. bell wire), connecting the centres of each jack to each other and the outsides to each other. The maximum amount of cable that can be reliably used is approximately 100 metres, however, the less the better.

Unfortunately the network hardware on the standard QL is not always reliable, but if you experience problems, it might be worth trying either a different QL or swapping the leads over. THOR XVIs should work without any problems, however, if you intend linking several QLs together, it is advisable to install Toolkit II on ROM (or SMSQ/E) in each of them (the memory only version of Toolkit II cannot improve the network due to timing). Minerva or SMSQ/E on all of the machines will also improve the Network handling.

Each computer (known as a station) must be given a separate station number so that it can be clearly identified within the Network. This station number is given to the machine when it is turned on (this will always be one) and so must be reset on each machine by the command NET.

### 52.1.2 A17.1.2 The Device Driver

Data is sent through the Network ports normally by using the NET device driver, which has the format:

NET<direction><station> (QL ROM)

NET<direction><station>_<buffer> (Toolkit II, THOR XVI)

The parameters of the NET device have the following meanings:

<direction>This is the direction in which data is to be sent. There is no default and this therefore must be one of the following letters:

- I - This is a listening station (input only)
- O - This is a transmitting station (output only)
- <station> This is the number of the station to communicate with and is in the form _n, where n represents the station number of the other machine you wish to communicate with.

  n=0 is treated as a special case - see below.

  The default is _0.

This Device Driver allows data to either be transmitted to a specific machine, or broadcast to any machine which is listening to the Network.

### 52.1.3 A17.1.3 Sending / Receiving Data

To broadcast data (sending it to all machines in the Network) you need to open a channel to NETO_0, for example:

```
OPEN_NEW #3,NETO_0
```

To listen to data which is being Broadcast, the listening stations will need to open a channel NETI_0, for example with:

```
OPEN_IN #4,NETI_0
```

If you open a net-in channel and specify your own station number, this is treated as a general listening station, allowing you to receive any data sent to that station by any machine in the Network. For example, you could use:

```
NET 12: OPEN_IN #4, NETI_12
```

to set up a general listening station.

If you want to open a net-in channel which will only listen for data from a specific machine in the Network, you will need to specify the station number of that machine, for example:

On station 3, enter:

```
NET 3: OPEN_NEW #3, NETO_12
```

Then on station 12, enter:

NET 12:OPEN_IN #3, NETI_3

will set up a link between stations 3 and 12 which only those machines can use. To allow station 12 to send data to station 3, you will also need to open a net-out channel, with:

```
OPEN_NEW #4, NETO_3
```

and to enable station 3 to listen to it, you could use on station 3:

```
OPEN_IN #4, NETI_3
```

(this allows station 3 to listen to any messages sent to it over the network by any other machine).

Each network channel can be input only or output only, thus bi-directional channels are not allowed. However, you can open as many channels as you like onto the Network on each machine, some of which may be output channels whilst others are input channels. If you try to send data down an input channel (or read data from an output channel), then a 'Bad Parameter' error will be returned.

Due to the way in which the data being sent over the Network ports is tested, you will need to open the output side of a Network link before the input side is opened, as otherwise, the two computers may miss each other's data header. If you later need to close the output port, you will need to inform the computers which are listening for data before the port is closed, as they themselves will need to close their input ports (any attempt to read any further data once the output port has been closed will result in an 'End of File' error). Only once the output port has been re-opened is it safe for them to re-open the input ports.

For this reason, if you plan to write a program which may open and close the output port several times in a session, it would be useful to have a secondary output port open at all times, which can be used to Broadcast to the listening stations when to open and/or close their own input ports.

Data is transmitted in packets of a specified size (the size depends on the type of device driver). The size of the packet determines the smallest amount of data that can be sent - any spare values will be set to zero. If the amount of data is greater than the packet size, then it will be sent as several packets. However, a packet will only be sent down the Network if it is full, therefore if some data remains to be sent which does not completely fill a packet, the sending machine will need to CLOSE the channel, or flush the

network (this requires a specialised routine - FLUSH will not work on a network device) in order to send the remaining pieces of data.

The NET device is greatly improved if Toolkit II is present (or a THOR XVI is being used), and we shall deal with this separately.

### 52.1.4 A17.1.4 QNet without Toolkit II

Data is transmitted in packets of 255 bytes preceded by a small Network header in the following format:

| Off-set | Name | Size | Description |
|---|---|---|---|
| 0 | NET.HE | byte | Destination station number (equivalent to NCIRIS on the Spectrum) |
| 1 | NET.SEI | byte | Number of sending station |
| 2 | NET.BL | byte | LSB of data block number |
| 3 | NET.BL | byte | MSB of data block number (note the reverse order because of the way in which words are stored for the Z80 on the Spectrum) |
| 4 | NET.TY | byte | Packet type: 0 data, 1 last block (EOF) |
| 5 | NET.NB | byte | Number of bytes in data block (0 to 255) |
| 6 | NET.DC | byte | Data checksum |
| 7 | NET.HC | byte | Header checksum |

The effects of the header depend on whether the sending machine is Broadcasting (ie. using NETO_0), in which case there is no handshaking, or not (in which case handshaking is enabled).

Before the packet is sent down the Network, the sending machine listens to the Network to check if it is being used. Once it is free, the sending machine then sends the header and if handshaking is enabled, waits for an acknowledgement from the destination machine that it is ready to receive.

Having received this acknowledgement (or if it is Broadcasting), the packet is sent. Once this is sent, if handshaking is enabled, then the sending machine again waits for the destination machine to acknowledge safe receipt; and if no such acknowledgement is received, tries all over again.

This means that no check is made on the data if the sending machine is Broadcasting, in which case it makes it very unreliable to Broadcast messages of more than 255 bytes (unless you have Toolkit II, a THOR XVI or Minerva; all of which improve the reliability, although it is still not 100%). This does also mean that if no stations are actually listening, the whole of the data will be lost.

When receiving data through the Network, the command EOF will only detect the end of the data if there are no more bytes to be read from the channel and the NET.TYPE in the header was set to EOF. The receiving machine will need to use the command PEND or EOFW to check if there is any data in the channel waiting to be read, unless you wish the program to just wait around for the data to be sent.

When the channel is closed, the device will try to output one final packet of data (this means that a minimum of one packet can be sent). If it fails to send the packet, then it will try a further 1399 times (causing an extremely long delay), after which the QL will give up. No error message is returned to tell the sending computer that it has failed to send the data. This means that CLOSEing a NETO channel, even though no data has been sent through the Network, produces an extremely long delay before the computer can do anything else (and may even crash some versions of the QL ROM if nothing has been written to the port - see CLOSE).

**Example**

```
COPY flp1_boot TO neto_2
```

copies the file flp1_boot to station 2.

### 52.1.5 A17.1.5 QNet Under Toolkit II

This is basically the same as the standard QNet driver, except that improvements have been made to improve handshaking and also to ensure that when an output channel is closed, whilst the driver keeps trying to send the last packet, the Break key is also checked for on the sending machine, allowing you to break into this early.

The Net header for the fileserver has also been improved to allow blocks of up to 1000 bytes to be sent at a time and also to improve the checksum.

If the driver fails to send the last packet (despite retrying 1399 times), or the Break key is detected, the message 'Net Aborted' is printed to #0 (although this does not stop the program), warning the user that the Network has failed.

The syntax has been extended to include a parameter <buffer> which represents the size of a buffer to be opened to receive bytes over the Network. It is in the form _n kilobytes and is really only applicable where the channel is NETI_0, as it specifies the size of the buffer (in kilobytes) to store the whole of a Broadcast message as it is transmitted. If no <buffer> is specified, it will use all but 2K of the free memory.

Toolkit II also implements a fileserver which allows a machine to directly access resources on another computer, by OPENing channels over the Network. Please refer to the FSERVE and NFS_USE keywords for details about the fileserver.

The MEM device can also be used to access another machine's resources over the Network. This is discussed in the Appendix on Device Drivers.

## 52.2 A17.2 Flexynet (DIY Toolkit - VOL X)

This can be used alongside the standard and Toolkit II QNet (subject to certain limitations - see below). The code will need to be loaded into either ROM (if you have an EPROM blower) or fast RAM (not the QL's internal 128K RAM - an expanded machine is therefore needed). Current versions of Flexynet will not work on machines which do not use a 68000 or 68008 chip (such as QXLs or Super Gold Cards), unless the cache has been disabled.

It has been implemented with a view to speeding up the transfer of data across the QNet, by allowing you to set the speed at which the data is to be transmitted (using the NETRATE command). The speed which the Networks will support depends upon the machines which are connected to QNet, with faster machines being able to receive data much faster than under the standard QNet (although transmission speed depends upon the speed of the machine at the other end of the QNet).

Although this can be used alongside QNet, there are really only two commands which allow you to send or receive multiple bytes sent over the network (NETSEND and NETREAD). There is currently no way of specifying which machine the data is to be sent to and therefore all machines in the network will be able to read the data sent.

You should not try to use both the standard QNet and Flexynet at the same time - we would recommend that a message is broadcast over the Network to all of the other machines first of all specifying that Flexynet is to be used and which machine is to receive the data and that the sending machine should then wait to hear that all other machines have closed down their network channels and that the receiving machine is ready to receive the data.

The commands NETBEEP and NETPOLL have also been added to allow the QL to use the Networks as a rudimentary form of digital sampling and even to generate sounds through the Network ports.

## 52.3 A17.3 Midinet

This extension provided with the Atari Emulators and SMSQ/E allows you to connect several ATARI computers in a Network by linking their MIDI ports together using suitable leads. As with QNet the machines must be arranged to form a complete circle with the MIDI OUT port of each machine being connected to the MIDI IN port of the next machine in the Network.

The Network will not work unless all machines are switched on (as with QNet) and unless all machines are running the MIDINET device driver, installed with LRESPR flp1_MIDINET_REXT (although the fileserver job need not be running except on the master machine).

Once connected, this system works very much in the same way as the QNet under Toolkit II, except that some file protection is provided to stop other machines on the Network accessing important files (see MIDINET).

The following commands are provided:

- MNET - Set the station number of this machine.
- MNET% - Return the station number.
- MNET_S% - Confirm whether a machine with a given station number is connected to the Network.
- MNET_ON - Switch on the device driver.
- MNET_OFF - Switch off the device driver (this allows the MIDI port to be used independently).
- MNET_USE - Change the letter which identifies the device driver (normally N).
- MIDINET - Start up the fileserver.

## 52.4 A17.4 Sernet

This extension provided with the Atari Emulators and SMSQ/E allows you to connect several different computers in a Network by linking their serial ports together using suitable leads.

This can therefore be used to connect all machines which currently are able to run QL software.

If you only have two machines in a Network, you can connect them by using a Null-Modem-Cable. However, with more than two machines, as with QNet the machines must be arranged to form a complete circle so that all of the output signals from one machine are connected to the input signals of the next machine.

The Network will not work unless all machines are switched on (as with QNet), all machines are running the SERNET device driver (installed with LRESPR flp1_SERNET_REXT) and SERNET has been configured on each machine to inform it which serial port it is to use for communications. Also, all of the machines must be set to the same BAUD rate before SERNET is loaded.

In order to improve the network, handshaking should be implemented on all ports, therefore to allow SERNET to use ser3 you may configure it to use: SER3hd (presuming hardware handshaking is available). If hardware handshaking is not available to some machines on the Network, you will need to use SER3xd on all the machines.

Once connected, this system works very much in the same way as the MIDINET.

The following commands are provided:

- SNET - Set the station number of this machine.
- SNET% - Returns the station number.
- SNET_S% - Confirm whether a machine with a given station number is connected to the Network.
- SNET_ROPEN - Re-opens the serial ports in case you have closed one from another program.
- SNET_USE - Change the letter which identifies the device driver (normally S).
- SERNET - Start up the fileserver.

## 52.5 A17.5 Amadeus Interlink

This is a box which can be linked to a QL or PC computer and allows you to connect up to 255 devices to a computer - these devices can be other Amadeus-fitted computers, printers or sound interfaces. If you use this to link computers together it provides in effect an extremely fast Network system, with more speed the faster the computer!

We do not have details of how the Network system works at present.

## 52.6 A17.6 QL - PC Fileserver

This is a software package which allows you (with the use of a cable which can be supplied) to link a QL to a PC computer via a free serial port and allows you to access the various devices provided by the PC. However, if you wish to use the package with Minerva, you will need at least v1.02 of the QL-PC Fileserver package. The original version will not work with the Super Gold Card and these users will need the QL-PC Fileserver II version (see below).

Basically, once the system has been set up and linked into both the QL and PC, you have to create a fileserver task on the PC by entering a new command on the PC (QLNET) and then on the QL side, simply set the correct BAUD rate (for the PC) and enter the command PCSERVE to inform the QL which serial port on the QL is linked to the PC.

Having done this, you are provided with various commands to find out details of the drives connected to the PC and can access them by simply using the device pcd, where DIR pcd1_ will provide a directory listing of drive A: on the PC, and DIR pcd3_ will provide a directory listing of drive C: on the PC (normally the hard disk).

Files can be saved onto the PC's devices in QL format to be read at a later stage by any other QL. You can also access the PC's printer (PC_DEV ser2,lpt1 redirects all output to ser2 to the lpt1 device on the PC) and even the PC's screen (the device PSCR is used to signify a channel is to be OPENed on the PC's screen, replacing the QL's SCR device).

Although speed is somewhat slower than using devices plugged into the QL, at least this means that you could get away without having to buy any disk drives (floppy or hard disk) for the QL.

One other thing that this package does is allow the QL to connect into a PC network, thus opening up the world of the PC in a cost-effective manner.

QL-PC Fileserver II is a newer version of the package which is much enhanced, allowing the QL to be connected to the PC via Amadeus Interlink as well as the serial ports. Full QL filename lengths are supported in this version and if you use the PC's screen to display the QL's output, this now supports QL windowing, colour and CON devices.

CHAPTER

FIFTYTHREE

# 18 CONFIGURING PROGRAMS

There are many different ways of allowing a user to configure a program. In this Appendix, we aim to highlight some of the options available, together with details of their benefits and pitfalls.

## 53.1 CONFIG Level 1 & Level 2

Many people who have used programs written for the Pointer Environment are used to using the Config program written by Qjump, or the MenuConfig program from Jochen Merz Software to configure programs.

These programs work by searching the program to be configured for a special 'config block', which has to have a special format.

If a program is written in 'C' or Machine Code, it is fairly easy to create a config block following the rules set out in the QPtr Toolkit manual.

However, many programmers find it much easier to create a program in compiled SuperBASIC. The only way of adding these config blocks to a compiled program is to use the Public Domain program BASCONFIG which can be linked into the program at compile time. This package creates a user defined config block which can be read using new SuperBASIC keywords …..

Unfortunately, BASCONFIG cannot be used with Turbo compiled programs as this does not allow toolkits to be linked to a compiled program.

Level 2 Config blocks allow automatic configuration of a new version of a program. The BASCONFIG package does not support these type of config blocks.

## 53.2 Passing Parameter with EXEC

If only one or two configuration options are required (such as where a program should look for a specific file), then it is sometimes easier to pass these parameters as an option string with the EXEC command (or similar).

The problem here is that it is all too easy to forget to include this string when starting up a program and many programs allow a long complicated list of parameters to be set in this way.

The option string can be read with functions such as OPTION_CMD$ or CMD$.

## 53.3 Making the configuration part of the program.

This means that the program must be loaded to be configured and then possibly reloaded into the machine for the configuration to take effect.

This makes the program bigger for something that may only be used once.

It is also more difficult to have several configurations of the same program.

## 53.4 Using a separate configuration file.

The problem here is how do you tell the main program where to load the configuration file from!

## 53.5 Using Environment Variables

This is OK, but user may forget to set the initial Environment Variable.

## 53.6 DATA_USE etc

The problem is that they can be altered externally by other programs. Only one setting at a time is possible.

- search