axiom™



The 30 Year Horizon

Manuel Bronstein *William Burge* *Timothy Daly*
James Davenport *Michael Dewar* *Martin Dunstan*
Albrecht Fortenbacher *Patrizia Gianni* *Johannes Grabmeier*
Jocelyn Guidry *Richard Jenks* *Larry Lambe*
Michael Monagan *Scott Morrison* *William Sit*
Jonathan Steinbach *Robert Sutor* *Barry Trager*
Stephen Watt *Jim Wen* *Clifton Williamson*

Volume 5: Axiom Interpreter

Contents

1 Credits **1**

1.0.1 defvar $credits 1

2 The Interpreter **5**

3 The Fundamental Data Structures **7**

3.1 The global variables 7

3.1.1 defvar $current-directory 7

3.1.2 defvar $current-directory 7

3.1.3 defvar $defaultMsgDatabaseName 8

3.1.4 defvar $defaultMsgDatabaseName 8

3.1.5 defvar $directory-list 8

3.1.6 defvar $directory-list 8

3.1.7 defvar $InitialModemapFrame 9

3.1.8 defvar $InitialModemapFrame 9

3.1.9 defvar $library-directory-list 9

3.1.10 defvar $library-directory-list 9

3.1.11 defvar $msgDatabaseName 9

3.1.12 defvar $msgDatabaseName 10

3.1.13 defvar $openServerIfTrue 10

3.1.14 defvar $openServerIfTrue 10

3.1.15 defvar $relative-directory-list 10

3.1.16 defvar $relative-directory-list 11

3.1.17 defvar $relative-library-directory-list 11

3.1.18 defvar $relative-library-directory-list 11

3.1.19 defvar $spadroot 11

3.1.20 defvar $spadroot 12

3.1.21 defvar $SpadServer 12

3.1.22 defvar $SpadServer 12

3.1.23 defvar $SpadServerName 12

3.1.24 defvar $SpadServerName 13

4 Starting Axiom **15**
4.1 Variables Used . . . 15
4.2 Data Structures . . . 15
4.3 Functions . . . 15
4.3.1 Set the restart hook . . . 15
4.3.2 restart function (The restart function) . . . 16
4.3.3 defun Non-interactive restarts . . . 18
4.3.4 defun The startup banner messages . . . 19
4.3.5 defun Make a vector of filler characters . . . 20
4.3.6 Starts the interpreter but do not read in profiles . . . 20
4.3.7 defvar $quitTag . . . 20
4.3.8 defun runspad . . . 21
4.3.9 defun Reset the stack limits . . . 21

5 Handling Terminal Input **23**
5.1 Streams . . . 23
5.1.1 defvar $curinstream . . . 23
5.1.2 defvar $curoutstream . . . 23
5.1.3 defvar $errorinstream . . . 23
5.1.4 defvar $erroroutstream . . . 24
5.1.5 defvar $*eof* . . . 24
5.1.6 defvar $*whitespace* . . . 24
5.1.7 defvar $InteractiveMode . . . 24
5.1.8 defvar $boot . . . 25
5.1.9 Top-level read-parse-eval-print loop . . . 25
5.1.10 defun ncIntLoop . . . 25
5.1.11 defvar $intTopLevel . . . 26
5.1.12 defvar $intRestart . . . 26
5.1.13 defun intloop . . . 26
5.1.14 defvar $ncMsgList . . . 27
5.1.15 defun SpadInterpretStream . . . 27
5.1.16 defvar $promptMsg . . . 28
5.1.17 defun GCL cmpnote function . . . 28
5.1.18 defvar $newcompErrorCount . . . 28
5.1.19 defvar $nopos . . . 28
5.2 The Read-Eval-Print Loop . . . 30
5.2.1 defun intloopReadConsole . . . 30
5.3 Helper Functions . . . 31
5.3.1 Get the value of an evironment variable . . . 31
5.3.2 defvar $intCoerceFailure . . . 32
5.3.3 defvar $intSpadReader . . . 32
5.3.4 defun InterpExecuteSpadSystemCommand . . . 32
5.3.5 defun ExecuteInterpSystemCommand . . . 33
5.3.6 defun Handle Synonyms . . . 33
5.3.7 defun Synonym File Reader . . . 33
5.3.8 defun init-memory-config . . . 34

5.3.9 Set spadroot to be the AXIOM shell variable . . . 35
5.3.10 Does the string start with this prefix? . . . 36
5.3.11 defun Interpret a line of lisp code . . . 36
5.3.12 Get the current directory . . . 36
5.3.13 Prepend the absolute path to a filename . . . 36
5.3.14 Make the initial modemap frame . . . 37
5.3.15 defun ncloopEscaped . . . 37
5.3.16 defun intloopProcessString . . . 37
5.3.17 defun ncloopParse . . . 38
5.3.18 defun next . . . 38
5.3.19 defun next1 . . . 38
5.3.20 defun incString . . . 39
5.3.21 Call the garbage collector . . . 39
5.3.22 defun reroot . . . 40
5.3.23 defun setCurrentLine . . . 41
5.3.24 Show the Axiom prompt . . . 42
5.3.25 defvar $frameAlist . . . 43
5.3.26 defvar $frameNumber . . . 43
5.3.27 defvar $currentFrameNum . . . 43
5.3.28 defvar $EndServerSession . . . 43
5.3.29 defvar $NeedToSignalSessionManager . . . 44
5.3.30 defvar $sockBufferLength . . . 44
5.3.31 READ-LINE in an Axiom server system . . . 44
5.3.32 defun protectedEVAL . . . 47
5.3.33 defvar $QuietCommand . . . 47
5.3.34 defun executeQuietCommand . . . 47
5.3.35 defun parseAndInterpret . . . 48
5.3.36 defun parseFromString . . . 48
5.3.37 defvar $interpOnly . . . 49
5.3.38 defvar $minivectorNames . . . 49
5.3.39 defvar $domPvar . . . 49
5.3.40 defun processInteractive . . . 49
5.3.41 defvar $ProcessInteractiveValue . . . 52
5.3.42 defvar $HTCompanionWindowID . . . 52
5.3.43 defun processInteractive1 . . . 52
5.3.44 defun interpretTopLevel . . . 53
5.3.45 defvar $genValue . . . 53
5.3.46 defun Type analyzes and evaluates expression x, returns object . . . 54
5.3.47 defun Dispatcher for the type analysis routines . . . 54
5.3.48 defun interpret2 . . . 55
5.3.49 defun Result Output Printing . . . 56
5.3.50 defun printStatisticsSummary . . . 57
5.3.51 defun printStorage . . . 58
5.3.52 defun printTypeAndTime . . . 58
5.3.53 defun printTypeAndTimeNormal . . . 59
5.3.54 defun printTypeAndTimeSaturn . . . 60

5.3.55 defun printAsTeX . . . 61
5.3.56 defun sameUnionBranch . . . 61
5.3.57 defun msgText . . . 61
5.3.58 defun Right-justify the Type output . . . 62
5.3.59 defun Destructively fix quotes in strings . . . 62
5.3.60 Include a file into the stream . . . 63
5.3.61 defun intloopInclude0 . . . 63
5.3.62 defun intloopProcess . . . 64
5.3.63 defun intloopSpadProcess . . . 64
5.3.64 defun intloopSpadProcess,interp . . . 65
5.3.65 defun phParse . . . 66
5.3.66 defun phIntReportMsgs . . . 66
5.3.67 defun phInterpret . . . 67
5.3.68 defun intInterpretPform . . . 67
5.3.69 defun zeroOneTran . . . 68
5.3.70 defun ncConversationPhase . . . 68
5.3.71 defun ncConversationPhase,wrapup . . . 68
5.3.72 defun ncError . . . 69
5.3.73 defun intloopEchoParse . . . 69
5.3.74 defun ncloopPrintLines . . . 70
5.3.75 defun mkLineList . . . 70
5.3.76 defun nonBlank . . . 71
5.3.77 defun ncloopDQlines . . . 71
5.3.78 defun poGlobalLinePosn . . . 72
5.3.79 defun streamChop . . . 72
5.3.80 defun ncloopInclude0 . . . 73
5.3.81 defun incStream . . . 73
5.3.82 defun incRenumber . . . 74
5.3.83 defun incZip . . . 74
5.3.84 defun incZip1 . . . 74
5.3.85 defun incIgen . . . 75
5.3.86 defun incIgen1 . . . 75
5.3.87 defun incRenumberLine . . . 75
5.3.88 defun incRenumberItem . . . 76
5.3.89 defun incHandleMessage . . . 76
5.3.90 defun incLude . . . 76
5.3.91 defmacro Rest . . . 77
5.3.92 defvar $Top . . . 77
5.3.93 defvar $IfSkipToEnd . . . 77
5.3.94 defvar $IfKeepPart . . . 77
5.3.95 defvar $IfSkipPart . . . 78
5.3.96 defvar $ElseifSkipToEnd . . . 78
5.3.97 defvar $ElseifKeepPart . . . 78
5.3.98 defvar $ElseifSkipPart . . . 78
5.3.99 defvar $ElseSkipToEnd . . . 78
5.3.100 defvar $ElseKeepPart . . . 79

5.3.101 defvar $Top? . . . 79
5.3.102 defvar $If? . . . 79
5.3.103 defvar $Elseif? . . . 79
5.3.104 defvar $Else? . . . 80
5.3.105 defvar $SkipEnd? . . . 80
5.3.106 defvar $KeepPart? . . . 80
5.3.107 defvar $SkipPart? . . . 81
5.3.108 defvar $Skipping? . . . 81
5.3.109 defun incLude1 . . . 81
5.3.110 defun xlPrematureEOF . . . 86
5.3.111 defun xlMsg . . . 86
5.3.112 defun xlOK . . . 86
5.3.113 defun xlOK1 . . . 86
5.3.114 defun incAppend . . . 87
5.3.115 defun incAppend1 . . . 87
5.3.116 defun incLine . . . 87
5.3.117 defun incLine1 . . . 88
5.3.118 defun inclmsgPrematureEOF . . . 88
5.3.119 defun theorigin . . . 88
5.3.120 defun porigin . . . 88
5.3.121 defun ifCond . . . 89
5.3.122 defun xlSkip . . . 89
5.3.123 defun xlSay . . . 89
5.3.124 defun inclmsgSay . . . 90
5.3.125 defun theid . . . 90
5.3.126 defun xlNoSuchFile . . . 90
5.3.127 defun inclmsgNoSuchFile . . . 91
5.3.128 defun thefname . . . 91
5.3.129 defun pfname . . . 91
5.3.130 defun xlCannotRead . . . 91
5.3.131 defun inclmsgCannotRead . . . 92
5.3.132 defun xlFileCycle . . . 92
5.3.133 defun inclmsgFileCycle . . . 92
5.3.134 defun xlConActive . . . 93
5.3.135 defun inclmsgConActive . . . 93
5.3.136 defun xlConStill . . . 94
5.3.137 defun inclmsgConStill . . . 94
5.3.138 defun xlConsole . . . 94
5.3.139 defun inclmsgConsole . . . 94
5.3.140 defun xlSkippingFin . . . 95
5.3.141 defun inclmsgFinSkipped . . . 95
5.3.142 defun xlPrematureFin . . . 95
5.3.143 defun inclmsgPrematureFin . . . 95
5.3.144 defun assertCond . . . 96
5.3.145 defun xlIfSyntax . . . 96
5.3.146 defun inclmsgIfSyntax . . . 97

5.3.147 defun xlIfBug . . . 97
5.3.148 defun inclmsgIfBug . . . 97
5.3.149 defun xlCmdBug . . . 98
5.3.150 defun inclmsgCmdBug . . . 98
5.3.151 defvar $incCommands . . . 98
5.3.152 defvar $pfMacros . . . 98
5.3.153 defun incClassify . . . 99
5.3.154 defun incCommand? . . . 100
5.3.155 defun incPrefix? . . . 100
5.3.156 defun incCommandTail . . . 101
5.3.157 defun incDrop . . . 101
5.3.158 defun inclFname . . . 102
5.3.159 defun incFileInput . . . 102
5.3.160 defun incConsoleInput . . . 102
5.3.161 defun incNConsoles . . . 103
5.3.162 defun incActive? . . . 103
5.3.163 defun incRgen . . . 103
5.3.164 defun Delay . . . 103
5.3.165 defvar $StreamNil . . . 104
5.3.166 defvar $StreamNil . . . 104
5.3.167 defun incRgen1 . . . 104

6 The Token Scanner 105
6.0.168 defvar $scanKeyWords . . . 105
6.0.169 defvar $infgeneric . . . 107
6.0.170 defun lineoftoks . . . 108
6.0.171 defun nextline . . . 110
6.0.172 defun scanIgnoreLine . . . 110
6.0.173 defun constoken . . . 111
6.0.174 defun scanToken . . . 111
6.0.175 defun lfid . . . 112
6.0.176 defun startsComment? . . . 113
6.0.177 defun scanComment . . . 113
6.0.178 defun lfcomment . . . 114
6.0.179 defun startsNegComment? . . . 114
6.0.180 defun scanNegComment . . . 114
6.0.181 defun lfnegcomment . . . 115
6.0.182 defun punctuation? . . . 115
6.0.183 defun scanPunct . . . 115
6.0.184 defun subMatch . . . 116
6.0.185 defun substringMatch . . . 116
6.0.186 defun scanKeyTr . . . 117
6.0.187 defun keyword . . . 118
6.0.188 defun keyword? . . . 118
6.0.189 defun scanPossFloat . . . 118
6.0.190 defun digit? . . . 119

6.0.191 defun lfkey . . . 119
6.0.192 defun spleI . . . 119
6.0.193 defun spleI1 . . . 120
6.0.194 defun scanEsc . . . 120
6.0.195 defvar $scanCloser . . . 122
6.0.196 defun scanCloser? . . . 123
6.0.197 defun scanWord . . . 123
6.0.198 defun scanExponent . . . 124
6.0.199 defun lffloat . . . 125
6.0.200 defmacro idChar? . . . 125
6.0.201 defun scanW . . . 125
6.0.202 defun posend . . . 126
6.0.203 defun scanSpace . . . 127
6.0.204 defun lfspaces . . . 127
6.0.205 defun scanString . . . 127
6.0.206 defun lfstring . . . 128
6.0.207 defun scanS . . . 128
6.0.208 defun scanTransform . . . 129
6.0.209 defun scanNumber . . . 129
6.0.210 defun rdigit? . . . 131
6.0.211 defun lfinteger . . . 131
6.0.212 defun lfrinteger . . . 131
6.0.213 defun scanCheckRadix . . . 131
6.0.214 defun scanEscape . . . 132
6.0.215 defun scanError . . . 132
6.0.216 defun lferror . . . 133
6.0.217 defvar $scanKeyTable . . . 133
6.0.218 defun scanKeyTableCons . . . 133
6.0.219 defvar $scanDict . . . 134
6.0.220 defun scanDictCons . . . 134
6.0.221 defun scanInsert . . . 135
6.0.222 defvar $scanPun . . . 136
6.0.223 defun scanPunCons . . . 137

7 Input Stream Parser **139**
7.0.224 defun Input Stream Parser . . . 139
7.0.225 defun npItem . . . 140
7.0.226 defun npItem1 . . . 140
7.0.227 defun npFirstTok . . . 141
7.0.228 defun Push one item onto $stack . . . 141
7.0.229 defun Pop one item off $stack . . . 142
7.0.230 defun Pop the second item off $stack . . . 142
7.0.231 defun Pop the third item off $stack . . . 142
7.0.232 defun npQualDef . . . 143
7.0.233 defun Advance over a keyword . . . 143
7.0.234 defun Advance the input stream . . . 143

7.0.235 defun npComma . . . 144
7.0.236 defun npTuple . . . 144
7.0.237 defun npCommaBackSet . . . 144
7.0.238 defun npQualifiedDefinition . . . 145
7.0.239 defun npQualified . . . 145
7.0.240 defun npDefinitionOrStatement . . . 145
7.0.241 defun npBackTrack . . . 146
7.0.242 defun npGives . . . 146
7.0.243 defun npLambda . . . 146
7.0.244 defun npType . . . 147
7.0.245 defun npMatch . . . 148
7.0.246 defun npSuch . . . 148
7.0.247 defun npWith . . . 148
7.0.248 defun npCompMissing . . . 149
7.0.249 defun npMissing . . . 149
7.0.250 defun npRestore . . . 150
7.0.251 defun Peek for keyword s, no advance of token stream . . . 150
7.0.252 defun npCategoryL . . . 150
7.0.253 defun npCategory . . . 151
7.0.254 defun npSCategory . . . 151
7.0.255 defun npSignature . . . 152
7.0.256 defun npSigItemlist . . . 152
7.0.257 defun npListing . . . 153
7.0.258 defun Always produces a list, fn is applied to it . . . 153
7.0.259 defun npSigItem . . . 154
7.0.260 defun npTypeVariable . . . 154
7.0.261 defun npSignatureDefinee . . . 154
7.0.262 defun npTypeVariablelist . . . 155
7.0.263 defun npSigDecl . . . 155
7.0.264 defun npPrimary . . . 155
7.0.265 defun npPrimary2 . . . 156
7.0.266 defun npADD . . . 156
7.0.267 defun npAdd . . . 157
7.0.268 defun npAtom2 . . . 157
7.0.269 defun npInfixOperator . . . 158
7.0.270 defun npInfixOp . . . 159
7.0.271 defun npPrefixColon . . . 159
7.0.272 defun npApplication . . . 160
7.0.273 defun npDotted . . . 160
7.0.274 defun npAnyNo . . . 160
7.0.275 defun npSelector . . . 161
7.0.276 defun npApplication2 . . . 161
7.0.277 defun npPrimary1 . . . 162
7.0.278 defun npMacro . . . 162
7.0.279 defun npMdef . . . 162
7.0.280 defun npMDEF . . . 163

7.0.281 defun npMDEFinition . . . 163
7.0.282 defun npFix . . . 164
7.0.283 defun npLet . . . 164
7.0.284 defun npLetQualified . . . 164
7.0.285 defun npDefinition . . . 165
7.0.286 defun npDefinitionItem . . . 165
7.0.287 defun npTyping . . . 166
7.0.288 defun npDefaultItemlist . . . 166
7.0.289 defun npSDefaultItem . . . 167
7.0.290 defun npDefaultItem . . . 167
7.0.291 defun npDefaultDecl . . . 168
7.0.292 defun npStatement . . . 168
7.0.293 defun npExport . . . 169
7.0.294 defun npLocalItemlist . . . 169
7.0.295 defun npSLocalItem . . . 170
7.0.296 defun npLocalItem . . . 170
7.0.297 defun npLocalDecl . . . 170
7.0.298 defun npLocal . . . 171
7.0.299 defun npFree . . . 171
7.0.300 defun npInline . . . 172
7.0.301 defun npIterate . . . 172
7.0.302 defun npBreak . . . 172
7.0.303 defun npLoop . . . 173
7.0.304 defun npIterators . . . 173
7.0.305 defun npIterator . . . 174
7.0.306 defun npSuchThat . . . 174
7.0.307 defun Apply argument 0 or more times . . . 175
7.0.308 defun npWhile . . . 175
7.0.309 defun npForIn . . . 175
7.0.310 defun npReturn . . . 176
7.0.311 defun npVoid . . . 177
7.0.312 defun npExpress . . . 177
7.0.313 defun npExpress1 . . . 177
7.0.314 defun npConditionalStatement . . . 178
7.0.315 defun npImport . . . 178
7.0.316 defun npQualTypelist . . . 178
7.0.317 defun npSQualTypelist . . . 179
7.0.318 defun npQualType . . . 179
7.0.319 defun npAndOr . . . 179
7.0.320 defun npEncAp . . . 180
7.0.321 defun npEncl . . . 180
7.0.322 defun npAtom1 . . . 181
7.0.323 defun npPDefinition . . . 181
7.0.324 defun npDollar . . . 181
7.0.325 defun npConstTok . . . 182
7.0.326 defun npBDefinition . . . 183

7.0.327 defun npBracketed . . . 183
7.0.328 defun npParened . . . 183
7.0.329 defun npBracked . . . 184
7.0.330 defun npBraced . . . 184
7.0.331 defun npAngleBared . . . 184
7.0.332 defun npDefn . . . 185
7.0.333 defun npDef . . . 185
7.0.334 defun npBPileDefinition . . . 186
7.0.335 defun npPileBracketed . . . 186
7.0.336 defun npPileDefinitionlist . . . 187
7.0.337 defun npListAndRecover . . . 187
7.0.338 defun npRecoverTrap . . . 188
7.0.339 defun npMoveTo . . . 189
7.0.340 defun syIgnoredFromTo . . . 189
7.0.341 defun syGeneralErrorHere . . . 190
7.0.342 defun sySpecificErrorHere . . . 190
7.0.343 defun sySpecificErrorAtToken . . . 190
7.0.344 defun npDefinitionlist . . . 191
7.0.345 defun npSemiListing . . . 191
7.0.346 defun npSemiBackSet . . . 191
7.0.347 defun npRule . . . 191
7.0.348 defun npSingleRule . . . 192
7.0.349 defun npDefTail . . . 192
7.0.350 defun npDefaultValue . . . 192
7.0.351 defun npWConditional . . . 193
7.0.352 defun npConditional . . . 193
7.0.353 defun npElse . . . 194
7.0.354 defun npBacksetElse . . . 195
7.0.355 defun npLogical . . . 195
7.0.356 defun npDisjand . . . 195
7.0.357 defun npDiscrim . . . 195
7.0.358 defun npQuiver . . . 196
7.0.359 defun npRelation . . . 196
7.0.360 defun npSynthetic . . . 196
7.0.361 defun npBy . . . 197
7.0.362 defun . . . 197
7.0.363 defun npSegment . . . 198
7.0.364 defun npArith . . . 198
7.0.365 defun npSum . . . 199
7.0.366 defun npTerm . . . 199
7.0.367 defun npRemainder . . . 199
7.0.368 defun npProduct . . . 200
7.0.369 defun npPower . . . 200
7.0.370 defun npAmpersandFrom . . . 200
7.0.371 defun npFromdom . . . 200
7.0.372 defun npFromdom1 . . . 201

7.0.373 defun npAmpersand . . . 202
7.0.374 defun npName . . . 202
7.0.375 defvar $npPParg . . . 202
7.0.376 defun npId . . . 202
7.0.377 defun npSymbolVariable . . . 203
7.0.378 defun npRightAssoc . . . 204
7.0.379 defun p o p o p o p = (((p o p) o p) o p) . . . 204
7.0.380 defun npInfGeneric . . . 205
7.0.381 defun npDDInfKey . . . 206
7.0.382 defun npInfKey . . . 206
7.0.383 defun npPushId . . . 207
7.0.384 defvar $npPParg . . . 207
7.0.385 defun npPP . . . 207
7.0.386 defun npPPff . . . 208
7.0.387 defun npPPg . . . 208
7.0.388 defun npPPf . . . 209
7.0.389 defun npEnclosed . . . 209
7.0.390 defun npState . . . 210
7.0.391 defun npTrap . . . 210
7.0.392 defun npTrapForm . . . 210
7.0.393 defun npVariable . . . 211
7.0.394 defun npVariablelist . . . 211
7.0.395 defun npVariableName . . . 211
7.0.396 defun npDecl . . . 212
7.0.397 defun npParenthesized . . . 212
7.0.398 defun npParenthesize . . . 213
7.0.399 defun npMissingMate . . . 213
7.0.400 defun npExit . . . 213
7.0.401 defun npPileExit . . . 214
7.0.402 defun npAssign . . . 214
7.0.403 defun npAssignment . . . 215
7.0.404 defun npAssignVariable . . . 215
7.0.405 defun npColon . . . 215
7.0.406 defun npTagged . . . 216
7.0.407 defun npTypedForm1 . . . 216
7.0.408 defun npTypified . . . 216
7.0.409 defun npTypeStyle . . . 217
7.0.410 defun npPretend . . . 217
7.0.411 defun npColonQuery . . . 217
7.0.412 defun npCoerceTo . . . 218
7.0.413 defun npTypedForm . . . 218
7.0.414 defun npRestrict . . . 218
7.0.415 defun npListofFun . . . 219
7.1 Macro handling . . . 219
7.1.1 defun phMacro . . . 219
7.1.2 defun macroExpanded . . . 220

7.1.3 defun macExpand . . . 220
7.1.4 defun macApplication . . . 221
7.1.5 defun mac0MLambdaApply . . . 221
7.1.6 defun mac0ExpandBody . . . 222
7.1.7 defun mac0InfiniteExpansion . . . 223
7.1.8 defun mac0InfiniteExpansion,name . . . 224
7.1.9 defun mac0GetName . . . 224
7.1.10 defun macId . . . 225
7.1.11 defun mac0Get . . . 226
7.1.12 defun macWhere . . . 226
7.1.13 defun macWhere,mac . . . 226
7.1.14 defun macLambda . . . 226
7.1.15 defun macLambda,mac . . . 227
7.1.16 defun Add appropriate definition the a Macro pform . . . 227
7.1.17 defun Add a macro to the global pfMacros list . . . 228
7.1.18 defun macSubstituteOuter . . . 228
7.1.19 defun mac0SubstituteOuter . . . 229
7.1.20 defun macLambdaParameterHandling . . . 229
7.1.21 defun macSubstituteId . . . 230

8 Pftrees 231
8.1 Abstract Syntax Trees Overview . . . 231
8.2 Structure handlers . . . 233
8.2.1 defun pfGlobalLinePosn . . . 233
8.2.2 defun pfCharPosn . . . 233
8.2.3 defun pfLinePosn . . . 233
8.2.4 defun pfFileName . . . 234
8.2.5 defun pfCopyWithPos . . . 234
8.2.6 defun pfMapParts . . . 234
8.2.7 defun pf0ApplicationArgs . . . 235
8.2.8 defun pf0FlattenSyntacticTuple . . . 235
8.2.9 defun pfSourcePosition . . . 236
8.2.10 defun Convert a Sequence node to a list . . . 236
8.2.11 defun pfSpread . . . 237
8.2.12 defun Deconstruct nodes to lists . . . 237
8.2.13 defun pfCheckMacroOut . . . 238
8.2.14 defun pfCheckArg . . . 239
8.2.15 defun pfCheckId . . . 239
8.2.16 defun pfFlattenApp . . . 239
8.2.17 defun pfCollect1? . . . 240
8.2.18 defun pfCollectVariable1 . . . 240
8.2.19 defun pfPushMacroBody . . . 241
8.2.20 defun pfSourceStok . . . 241
8.2.21 defun pfTransformArg . . . 242
8.2.22 defun pfTaggedToTyped1 . . . 242
8.2.23 defun pfSuch . . . 242

8.3 Special Nodes . . . 243
8.3.1 defun Create a Listof node . . . 243
8.3.2 defun pfNothing . . . 243
8.3.3 defun Is this a Nothing node? . . . 243
8.4 Leaves . . . 244
8.4.1 defun Create a Document node . . . 244
8.4.2 defun Construct an Id node . . . 244
8.4.3 defun Is this an Id node? . . . 244
8.4.4 defun Construct an Id leaf node . . . 244
8.4.5 defun Return the Id part . . . 245
8.4.6 defun Construct a Leaf node . . . 245
8.4.7 defun Is this a leaf node? . . . 245
8.4.8 defun Return the token position of a leaf node . . . 246
8.4.9 defun Return the Leaf Token . . . 246
8.4.10 defun Is this a Literal node? . . . 246
8.4.11 defun Create a LiteralClass node . . . 246
8.4.12 defun Return the LiteralString . . . 247
8.4.13 defun Return the parts of a tree node . . . 247
8.4.14 defun Return the argument unchanged . . . 247
8.4.15 defun pfPushBody . . . 247
8.4.16 defun An S-expression which people can read. . . . 248
8.4.17 defun Create a human readable S-expression . . . 248
8.4.18 defun Construct a Symbol or Expression node . . . 249
8.4.19 defun Construct a Symbol leaf node . . . 249
8.4.20 defun Is this a Symbol node? . . . 250
8.4.21 defun Return the Symbol part . . . 250
8.5 Trees . . . 250
8.5.1 defun Construct a tree node . . . 250
8.5.2 defun Construct an Add node . . . 250
8.5.3 defun Construct an And node . . . 251
8.5.4 defun pfAttribute . . . 251
8.5.5 defun Return an Application node . . . 251
8.5.6 defun Return the Arg part of an Application node . . . 252
8.5.7 defun Return the Op part of an Application node . . . 252
8.5.8 defun Is this an And node? . . . 252
8.5.9 defun Return the Left part of an And node . . . 252
8.5.10 defun Return the Right part of an And node . . . 253
8.5.11 defun Flatten a list of lists . . . 253
8.5.12 defun Is this an Application node? . . . 253
8.5.13 defun Create an Assign node . . . 253
8.5.14 defun Is this an Assign node? . . . 254
8.5.15 defun Return the parts of an LhsItem of an Assign node . . . 254
8.5.16 defun Return the LhsItem of an Assign node . . . 254
8.5.17 defun Return the RHS of an Assign node . . . 254
8.5.18 defun Construct an application node for a brace . . . 255
8.5.19 defun Construct an Application node for brace-bars . . . 255

8.5.20 defun Construct an Application node for a bracket 255
8.5.21 defun Construct an Application node for bracket-bars 255
8.5.22 defun Create a Break node 256
8.5.23 defun Is this a Break node? 256
8.5.24 defun Return the From part of a Break node 256
8.5.25 defun Construct a Coerceto node 257
8.5.26 defun Is this a CoerceTo node? 257
8.5.27 defun Return the Expression part of a CoerceTo node 257
8.5.28 defun Return the Type part of a CoerceTo node 257
8.5.29 defun Return the Body of a Collect node 258
8.5.30 defun Return the Iterators of a Collect node 258
8.5.31 defun Create a Collect node 258
8.5.32 defun Is this a Collect node? 258
8.5.33 defun pfDefinition 259
8.5.34 defun Return the Lhs of a Definition node 259
8.5.35 defun Return the Rhs of a Definition node 259
8.5.36 defun Is this a Definition node? 259
8.5.37 defun Return the parts of a Definition node 260
8.5.38 defun Create a Do node 260
8.5.39 defun Is this a Do node? 260
8.5.40 defun Return the Body of a Do node 260
8.5.41 defun Construct a Sequence node 261
8.5.42 defun Construct an Exit node 261
8.5.43 defun Is this an Exit node? 261
8.5.44 defun Return the Cond part of an Exit 261
8.5.45 defun Return the Expression part of an Exit 262
8.5.46 defun Create an Export node 262
8.5.47 defun Construct an Expression leaf node 262
8.5.48 defun pfFirst 262
8.5.49 defun Create an Application Fix node 263
8.5.50 defun Create a Free node 263
8.5.51 defun Is this a Free node? 263
8.5.52 defun Return the parts of the Items of a Free node 264
8.5.53 defun Return the Items of a Free node 264
8.5.54 defun Construct a Forin node 264
8.5.55 defun Is this a ForIn node? 264
8.5.56 defun Return all the parts of the LHS of a ForIn node 265
8.5.57 defun Return the LHS part of a ForIn node 265
8.5.58 defun Return the Whole part of a ForIn node 265
8.5.59 defun pfFromDom 265
8.5.60 defun Construct a Fromdom node 266
8.5.61 defun Is this a Fromdom mode? 266
8.5.62 defun Return the What part of a Fromdom node 266
8.5.63 defun Return the Domain part of a Fromdom node 267
8.5.64 defun Construct a Hide node 267
8.5.65 defun pfIf 267

8.5.66 defun Is this an If node? . . . 267
8.5.67 defun Return the Cond part of an If . . . 268
8.5.68 defun Return the Then part of an If . . . 268
8.5.69 defun pfIfThenOnly . . . 268
8.5.70 defun Return the Else part of an If . . . 268
8.5.71 defun Construct an Import node . . . 269
8.5.72 defun Construct an Iterate node . . . 269
8.5.73 defun Is this an Iterate node? . . . 269
8.5.74 defun Handle an infix application . . . 269
8.5.75 defun Create an Inline node . . . 270
8.5.76 defun pfLam . . . 270
8.5.77 defun pfLambda . . . 271
8.5.78 defun Return the Body part of a Lambda node . . . 271
8.5.79 defun Return the Rets part of a Lambda node . . . 271
8.5.80 defun Is this a Lambda node? . . . 271
8.5.81 defun Return the Args part of a Lambda node . . . 272
8.5.82 defun Return the Args of a Lambda Node . . . 272
8.5.83 defun Construct a Local node . . . 272
8.5.84 defun Is this a Local node? . . . 272
8.5.85 defun Return the parts of Items of a Local node . . . 273
8.5.86 defun Return the Items of a Local node . . . 273
8.5.87 defun Construct a Loop node . . . 273
8.5.88 defun pfLoop1 . . . 273
8.5.89 defun Is this a Loop node? . . . 274
8.5.90 defun Return the Iterators of a Loop node . . . 274
8.5.91 defun pf0LoopIterators . . . 274
8.5.92 defun pfLp . . . 274
8.5.93 defun Create a Macro node . . . 275
8.5.94 defun Is this a Macro node? . . . 275
8.5.95 defun Return the Lhs of a Macro node . . . 275
8.5.96 defun Return the Rhs of a Macro node . . . 275
8.5.97 defun Construct an MLambda node . . . 276
8.5.98 defun Is this an MLambda node? . . . 276
8.5.99 defun Return the Args of an MLambda . . . 276
8.5.100 defun Return the parts of an MLambda argument . . . 276
8.5.101 defun pfMLambdaBody . . . 277
8.5.102 defun Is this a Not node? . . . 277
8.5.103 defun Return the Arg part of a Not node . . . 277
8.5.104 defun Construct a NoValue node . . . 277
8.5.105 defun Is this a Novalue node? . . . 278
8.5.106 defun Return the Expr part of a Novalue node . . . 278
8.5.107 defun Construct an Or node . . . 278
8.5.108 defun Is this an Or node? . . . 278
8.5.109 defun Return the Left part of an Or node . . . 279
8.5.110 defun Return the Right part of an Or node . . . 279
8.5.111 defun Return the part of a parenthesised expression . . . 279

8.5.112 defun pfPretend . . . 279
8.5.113 defun Is this a Pretend node? . . . 280
8.5.114 defun Return the Expression part of a Pretend node . . . 280
8.5.115 defun Return the Type part of a Pretend node . . . 280
8.5.116 defun Construct a QualType node . . . 280
8.5.117 defun Construct a Restrict node . . . 281
8.5.118 defun Is this a Restrict node? . . . 281
8.5.119 defun Return the Expr part of a Restrict node . . . 281
8.5.120 defun Return the Type part of a Restrict node . . . 281
8.5.121 defun Construct a RetractTo node . . . 282
8.5.122 defun Construct a Return node . . . 282
8.5.123 defun Is this a Return node? . . . 282
8.5.124 defun Return the Expr part of a Return node . . . 282
8.5.125 defun pfReturnNoName . . . 283
8.5.126 defun Construct a ReturnTyped node . . . 283
8.5.127 defun Construct a Rule node . . . 283
8.5.128 defun Return the Lhs of a Rule node . . . 284
8.5.129 defun Return the Rhs of a Rule node . . . 284
8.5.130 defun Is this a Rule node? . . . 284
8.5.131 defun pfSecond . . . 284
8.5.132 defun Construct a Sequence node . . . 285
8.5.133 defun Return the Args of a Sequence node . . . 285
8.5.134 defun Is this a Sequence node? . . . 285
8.5.135 defun Return the parts of the Args of a Sequence node . . . 285
8.5.136 defun Create a Suchthat node . . . 286
8.5.137 defun Is this a SuchThat node? . . . 286
8.5.138 defun Return the Cond part of a SuchThat node . . . 286
8.5.139 defun Create a Tagged node . . . 286
8.5.140 defun Is this a Tagged node? . . . 287
8.5.141 defun Return the Expression portion of a Tagged node . . . 287
8.5.142 defun Return the Tag of a Tagged node . . . 287
8.5.143 defun pfTaggedToTyped . . . 287
8.5.144 defun pfTweakIf . . . 288
8.5.145 defun Construct a Typed node . . . 288
8.5.146 defun Is this a Typed node? . . . 289
8.5.147 defun Return the Type of a Typed node . . . 289
8.5.148 defun Return the Id of a Typed node . . . 289
8.5.149 defun Construct a Typing node . . . 289
8.5.150 defun Return a Tuple node . . . 290
8.5.151 defun Return a Tuple from a List . . . 290
8.5.152 defun Is this a Tuple node? . . . 290
8.5.153 defun Return the Parts of a Tuple node . . . 291
8.5.154 defun Return the parts of a Tuple . . . 291
8.5.155 defun Return a list from a Sequence node . . . 291
8.5.156 defun The comment is attached to all signatutres . . . 291
8.5.157 defun Construct a WDeclare node . . . 292

8.5.158 defun Construct a Where node . . . 292
8.5.159 defun Is this a Where node? . . . 292
8.5.160 defun Return the parts of the Context of a Where node . . . 293
8.5.161 defun Return the Context of a Where node . . . 293
8.5.162 defun Return the Expr part of a Where node . . . 293
8.5.163 defun Construct a While node . . . 293
8.5.164 defun Is this a While node? . . . 294
8.5.165 defun Return the Cond part of a While node . . . 294
8.5.166 defun Construct a With node . . . 294
8.5.167 defun Create a Wrong node . . . 294
8.5.168 defun Is this a Wrong node? . . . 295

9 Pftree to s-expression translation 297
9.0.169 defun Pftree to s-expression translation . . . 297
9.0.170 defun Pftree to s-expression translation inner function . . . 298
9.0.171 defun Convert a Literal to an S-expression . . . 302
9.0.172 defun Convert a float to an S-expression . . . 303
9.0.173 defun Change an Application node to an S-expression . . . 303
9.0.174 defun Convert a SuchThat node to an S-expression . . . 305
9.0.175 defun pfOp2Sex . . . 306
9.0.176 defun pmDontQuote? . . . 307
9.0.177 defun hasOptArgs? . . . 307
9.0.178 defun Convert a Sequence node to an S-expression . . . 308
9.0.179 defun pfSequence2Sex0 . . . 308
9.0.180 defun Convert a loop node to an S-expression . . . 309
9.0.181 defun Change a Collect node to an S-expression . . . 312
9.0.182 defun Convert a Definition node to an S-expression . . . 313
9.0.183 defun Convert a Lambda node to an S-expression . . . 314
9.0.184 defun pfCollectArgTran . . . 315
9.0.185 defun Convert a Lambda node to an S-expression . . . 315
9.0.186 defun Convert a Rule node to an S-expression . . . 316
9.0.187 defun Convert the Lhs of a Rule to an S-expression . . . 316
9.0.188 defun Convert the Rhs of a Rule to an S-expression . . . 317
9.0.189 defun Convert a Rule predicate to an S-expression . . . 317
9.0.190 defun patternVarsOf . . . 319
9.0.191 defun patternVarsOf1 . . . 319
9.0.192 defun pvarPredTran . . . 320
9.0.193 defun Convert the Lhs of a Rule node to an S-expression . . . 320
9.0.194 defvar $dotdot . . . 321
9.0.195 defun Translate ops into internal symbols . . . 321

10 Keyed Message Handling 323
10.0.196 defvar $cacheMessages . . . 324
10.0.197 defvar $msgAlist . . . 324
10.0.198 defvar $msgDatabaseName . . . 324
10.0.199 defvar $testingErrorPrefix . . . 325

10.0.200 defvar $texFormatting . . . 325
10.0.201 defvar $*msghash* . . . 325
10.0.202 defvar $msgdbPrims . . . 325
10.0.203 defvar $msgdbPunct . . . 325
10.0.204 defvar $msgdbNoBlanksBeforeGroup . . . 326
10.0.205 defvar $msgdbNoBlanksAfterGroup . . . 326
10.0.206 defun Fetch a message from the message database . . . 326
10.0.207 defun Cache messages read from message database . . . 327
10.0.208 defun getKeyedMsg . . . 327
10.0.209 defun Say a message using a keyed lookup . . . 327
10.0.210 defun Handle msg formatting and print to file . . . 328
10.0.211 defun Break a message into words . . . 328
10.0.212 defun Write a msg into spadmsg.listing file . . . 329
10.0.213 defun sayMSG . . . 329

11 Stream Utilities **331**
11.0.214 defun npNull . . . 331
11.0.215 defun StreamNull . . . 331

12 Code Piles **333**
12.0.216 defun insertpile . . . 333
12.0.217 defun pilePlusComment . . . 334
12.0.218 defun pilePlusComments . . . 334
12.0.219 defun pileTree . . . 335
12.0.220 defun pileColumn . . . 335
12.0.221 defun pileForests . . . 335
12.0.222 defun pileForest . . . 336
12.0.223 defun pileForest1 . . . 336
12.0.224 defun eqpileTree . . . 337
12.0.225 defun pileCtree . . . 338
12.0.226 defun pileCforest . . . 338
12.0.227 defun enPile . . . 338
12.0.228 defun firstTokPosn . . . 339
12.0.229 defun lastTokPosn . . . 339
12.0.230 defun separatePiles . . . 339

13 Dequeue Functions **341**
13.0.231 defun dqUnit . . . 341
13.0.232 defun dqConcat . . . 341
13.0.233 defun dqAppend . . . 342
13.0.234 defun dqToList . . . 342

14 Message Handling **343**
14.1 The Line Object . . . 343
14.1.1 defun Line object creation . . . 343
14.1.2 defun Line element 0; Extra blanks . . . 343
14.1.3 defun Line element 1; String . . . 343
14.1.4 defun Line element 2; Globlal number . . . 344
14.1.5 defun Line element 2; Set Global number . . . 344
14.1.6 defun Line elemnt 3; Local number . . . 344
14.1.7 defun Line element 4; Place of origin . . . 344
14.1.8 defun Line element 4: Is it a filename? . . . 345
14.1.9 defun Line element 4: Is it a filename? . . . 345
14.1.10 defun Line element 4; Get filename . . . 345
14.2 Messages . . . 345
14.2.1 defun msgCreate . . . 345
14.2.2 defun getMsgPosTagOb . . . 346
14.2.3 defun getMsgKey . . . 346
14.2.4 defun getMsgArgL . . . 347
14.2.5 defun getMsgPrefix . . . 347
14.2.6 defun setMsgPrefix . . . 347
14.2.7 defun getMsgText . . . 347
14.2.8 defun setMsgText . . . 347
14.2.9 defun getMsgPrefix? . . . 348
14.2.10 defun getMsgTag . . . 348
14.2.11 defun getMsgTag? . . . 348
14.2.12 defun line? . . . 349
14.2.13 defun leader? . . . 349
14.2.14 defun toScreen? . . . 349
14.2.15 defun ncSoftError . . . 349
14.2.16 defun ncHardError . . . 350
14.2.17 defun desiredMsg . . . 350
14.2.18 defun processKeyedError . . . 351
14.2.19 defun msgOutputter . . . 351
14.2.20 defun listOutputter . . . 352
14.2.21 defun getStFromMsg . . . 352
14.2.22 defvar $preLength . . . 353
14.2.23 defun getPreStL . . . 353
14.2.24 defun getPosStL . . . 354
14.2.25 defun ppos . . . 355
14.2.26 defun remFile . . . 355
14.2.27 defun showMsgPos? . . . 355
14.2.28 defvar $imPrGuys . . . 356
14.2.29 defun msgImPr? . . . 356
14.2.30 defun getMsgCatAttr . . . 356
14.2.31 defun getMsgPos . . . 357
14.2.32 defun getMsgFTTag? . . . 357
14.2.33 defun decideHowMuch . . . 357

14.2.34 defun poNopos? . . . 358
14.2.35 defun poPosImmediate? . . . 358
14.2.36 defun poFileName . . . 358
14.2.37 defun poGetLineObject . . . 359
14.2.38 defun poLinePosn . . . 359
14.2.39 defun listDecideHowMuch . . . 359
14.2.40 defun remLine . . . 360
14.2.41 defun getMsgKey? . . . 360
14.2.42 defun getMsgLitSym . . . 360
14.2.43 defun tabbing . . . 360
14.2.44 defvar $toWhereGuys . . . 361
14.2.45 defun getMsgToWhere . . . 361
14.2.46 defun toFile? . . . 361
14.2.47 defun alreadyOpened? . . . 361
14.2.48 defun setMsgForcedAttrList . . . 362
14.2.49 defun setMsgForcedAttr . . . 362
14.2.50 defvar $attrCats . . . 362
14.2.51 defun whichCat . . . 363
14.2.52 defun setMsgCatlessAttr . . . 363
14.2.53 defun putDatabaseStuff . . . 363
14.2.54 defun getMsgInfoFromKey . . . 364
14.2.55 defun setMsgUnforcedAttrList . . . 364
14.2.56 defun setMsgUnforcedAttr . . . 365
14.2.57 defvar $imPrTagGuys . . . 365
14.2.58 defun initImPr . . . 365
14.2.59 defun initToWhere . . . 366
14.2.60 defun ncBug . . . 366
14.2.61 defun processMsgList . . . 367
14.2.62 defun erMsgSort . . . 367
14.2.63 defun erMsgCompare . . . 368
14.2.64 defun compareposns . . . 368
14.2.65 defun erMsgSep . . . 368
14.2.66 defun makeMsgFromLine . . . 369
14.2.67 defun rep . . . 369
14.2.68 defun getLinePos . . . 370
14.2.69 defun getLineText . . . 370
14.2.70 defun queueUpErrors . . . 370
14.2.71 defun thisPosIsLess . . . 372
14.2.72 defun thisPosIsEqual . . . 372
14.2.73 defun redundant . . . 372
14.2.74 defvar $repGuys . . . 373
14.2.75 defun msgNoRep? . . . 373
14.2.76 defun sameMsg? . . . 374
14.2.77 defun processChPosesForOneLine . . . 374
14.2.78 defun poCharPosn . . . 375
14.2.79 defun makeLeaderMsg . . . 375

14.2.80 defun posPointers . . . 376
14.2.81 defun getMsgPos2 . . . 376
14.2.82 defun insertPos . . . 377
14.2.83 defun putFTText . . . 377
14.2.84 defun From . . . 378
14.2.85 defun To . . . 378
14.2.86 defun FromTo . . . 378

15 The Interpreter Syntax **381**
15.1 syntax assignment . . . 381
15.2 syntax blocks . . . 384
15.3 system clef . . . 386
15.4 syntax collection . . . 387
15.5 syntax for . . . 389
15.6 syntax if . . . 393
15.7 syntax iterate . . . 395
15.8 syntax leave . . . 396
15.9 syntax parallel . . . 397
15.10 syntax repeat . . . 400
15.11 syntax suchthat . . . 404
15.12 syntax syntax . . . 405
15.13 syntax while . . . 405

16 Abstract Syntax Trees (ptrees) **409**
16.0.1 defun Construct a leaf token . . . 409
16.0.2 defun Return a part of a node . . . 410
16.0.3 defun Compare a part of a node . . . 410
16.0.4 defun pfNoPosition? . . . 410
16.0.5 defun poNoPosition? . . . 411
16.0.6 defun tokType . . . 411
16.0.7 defun tokPart . . . 411
16.0.8 defun tokPosn . . . 411
16.0.9 defun pfNoPosition . . . 412
16.0.10 defun poNoPosition . . . 412

17 Attributed Structures **413**
17.0.11 defun ncTag . . . 413
17.0.12 defun ncAlist . . . 413
17.0.13 defun ncEltQ . . . 414
17.0.14 defun ncPutQ . . . 414

18 Function Selection **417**
18.0.15 defun ofCategory . . . 417
18.0.16 defun isPartialMode . . . 418
18.0.17 defun hasCaty . . . 418
18.0.18 defun domArg . . . 420

18.0.19 defun domArg2 . . . 420
18.0.20 defun hasSig . . . 421
18.0.21 defun hasAtt . . . 422
18.0.22 defun hasSigAnd . . . 423
18.0.23 defun hasSigOr . . . 424
18.0.24 defun hasAttSig . . . 424
18.0.25 defun hasCate1 . . . 425
18.0.26 defun hasCatExpression . . . 425
18.0.27 defun unifyStruct . . . 426
18.0.28 defun unifyStructVar . . . 427
18.0.29 defun containsVars . . . 428
18.0.30 defun containsVars1 . . . 429
18.0.31 defun hasCaty1 . . . 429
18.0.32 defun mkDomPvar . . . 430
18.0.33 defun hasCate . . . 431
18.0.34 defun constructSubst . . . 432
18.0.35 defun hasCateSpecial . . . 432
18.0.36 defun hasCateSpecialNew . . . 433
18.0.37 defun defaultTargetFE . . . 435
18.0.38 defun isEqualOrSubDomain . . . 436

19 System Command Handling 437
19.1 Variables Used . . . 439
19.1.1 defvar $systemCommands . . . 439
19.1.2 defvar $syscommands . . . 440
19.1.3 defvar $noParseCommands . . . 440
19.2 Functions . . . 441
19.2.1 defun handleNoParseCommands . . . 441
19.2.2 defun Handle a top level command . . . 442
19.2.3 defun Split block into option block . . . 443
19.2.4 defun Tokenize a system command . . . 443
19.2.5 defun Handle system commands . . . 444
19.2.6 defun Select commands matching this user level . . . 444
19.2.7 defun No command begins with this string . . . 445
19.2.8 defun No option begins with this string . . . 445
19.2.9 defvar $oldline . . . 445
19.2.10 defun No command/option begins with this string . . . 445
19.2.11 defun Option not available at this user level . . . 446
19.2.12 defun Command not available at this user level . . . 446
19.2.13 defun Command not available error message . . . 446
19.2.14 defun satisfiesUserLevel . . . 447
19.2.15 defun hasOption . . . 447
19.2.16 defun terminateSystemCommand . . . 448
19.2.17 defun Terminate a system command . . . 448
19.2.18 defun commandAmbiguityError . . . 448
19.2.19 defun getParserMacroNames . . . 449

19.2.20 defun clearParserMacro . . . 449
19.2.21 defun displayMacro . . . 449
19.2.22 defun displayWorkspaceNames . . . 450
19.2.23 defun getWorkspaceNames . . . 451
19.2.24 defun fixObjectForPrinting . . . 452
19.2.25 defun displayProperties,sayFunctionDeps . . . 452
19.2.26 defun displayValue . . . 455
19.2.27 defun displayType . . . 456
19.2.28 defun getAndSay . . . 457
19.2.29 defun displayProperties . . . 457
19.2.30 defun displayParserMacro . . . 460
19.2.31 defun displayCondition . . . 461
19.2.32 defun interpFunctionDepAlists . . . 461
19.2.33 defun displayModemap . . . 462
19.2.34 defun displayMode . . . 462
19.2.35 defun Split into tokens delimted by spaces . . . 463
19.2.36 defun Convert string tokens to their proper type . . . 463
19.2.37 defun Is the argument string an integer? . . . 464
19.2.38 defun Handle parsed system commands . . . 464
19.2.39 defun Parse a system command . . . 465
19.2.40 defun Get first word in a string . . . 465
19.2.41 defun Unabbreviate keywords in commands . . . 465
19.2.42 defun The command is ambiguous error . . . 466
19.2.43 defun Remove the spaces surrounding a string . . . 467
19.2.44 defun Remove the lisp command prefix . . . 467
19.2.45 defun Handle the)lisp command . . . 468
19.2.46 defun The)boot command is no longer supported . . . 468
19.2.47 defun Handle the)system command . . . 468
19.2.48 defun Handle the)synonym command . . . 469
19.2.49 defun Handle the synonym system command . . . 469
19.2.50 defun printSynonyms . . . 470
19.2.51 defun Print a list of each matching synonym . . . 470
19.2.52 defvar $tokenCommands . . . 471
19.2.53 defvar $InitialCommandSynonymAlist . . . 472
19.2.54 defun Print the current version information . . . 472
19.2.55 defvar $CommandSynonymAlist . . . 474
19.2.56 defun ncloopCommand . . . 474
19.2.57 defun ncloopPrefix? . . . 475
19.2.58 defun selectOptionLC . . . 475
19.2.59 defun selectOption . . . 475

20)abbreviations help page Command 477
20.1 abbreviations help page man page . . . 477
20.2 Functions . . . 479
20.2.1 defun abbreviations . . . 479
20.2.2 defun abbreviationsSpad2Cmd . . . 479

20.2.3 defun listConstructorAbbreviations 480

21)boot help page Command 483
21.1 boot help page man page 483
21.2 Functions 484

22)browse help page Command 485
22.1 browse help page man page 485
22.2 Overview 485
22.3 Browsers, MathML, and Fonts 486
22.4 The axServer/multiServ loop 487
22.5 The)browse command 488
22.6 Variables Used 489
22.7 Functions 489
22.8 The server support code 489

23)cd help page Command 491
23.1 cd help page man page 491
23.2 Variables Used 492
23.3 Functions 492

24)clear help page Command 493
24.1 clear help page man page 493
24.2 Variables Used 495
24.2.1 defvar $clearOptions 495
24.3 Functions 495
24.3.1 defun clear 495
24.3.2 defvar $clearExcept 495
24.3.3 defun clearSpad2Cmd 496
24.3.4 defun clearCmdSortedCaches 497
24.3.5 defun compiledLookupCheck 497
24.3.6 defvar $functionTable 498
24.3.7 defun clearCmdCompletely 498
24.3.8 defun clearCmdAll 499
24.3.9 defun clearMacroTable 500
24.3.10 defun clearCmdExcept 500
24.3.11 defun clearCmdParts 501

25)close help page Command 505
25.1 close help page man page 505
25.2 Functions 506
25.2.1 defun queryClients 506
25.2.2 defun close 506

26)compile help page Command **509**
26.1 compile help page man page . . . 509
26.2 Functions . . . 511
26.2.1 defvar $/editfile . . . 511

27)copyright help page Command **513**
27.1 copyright help page man page . . . 513
27.2 Functions . . . 518
27.2.1 defun copyright . . . 518
27.2.2 defun trademark . . . 519

28)credits help page Command **521**
28.1 credits help page man page . . . 521
28.2 Variables Used . . . 521
28.3 Functions . . . 521
28.3.1 defun credits . . . 521

29)describe help page Command **523**
29.1 describe help page man page . . . 523
29.1.1 defvar $describeOptions . . . 524
29.2 Functions . . . 524
29.2.1 defun Print comment strings from algebra libraries . . . 524
29.2.2 defun describeSpad2Cmd . . . 524
29.2.3 defun cleanline . . . 525
29.2.4 defun flatten . . . 527

30)display help page Command **529**
30.1 display help page man page . . . 529
30.1.1 defvar $displayOptions . . . 531
30.2 Functions . . . 531
30.2.1 defun display . . . 531
30.2.2 displaySpad2Cmd . . . 531
30.2.3 defun abbQuery . . . 532
30.2.4 defun displayOperations . . . 533
30.2.5 defun yesanswer . . . 533
30.2.6 defun displayMacros . . . 534
30.2.7 defun sayExample . . . 535
30.2.8 defun cleanupLine . . . 536

31)edit help page Command **539**
31.1 edit help page man page . . . 539
31.2 Functions . . . 540
31.2.1 defun edit . . . 540
31.2.2 defun editSpad2Cmd . . . 540
31.2.3 defun Implement the)edit command . . . 541
31.2.4 defun updateSourceFiles . . . 542

32)fin help page Command **543**
32.1 fin help page man page . . . 543
32.1.1 defun Exit from the interpreter to lisp . . . 544
32.2 Functions . . . 544

33)frame help page Command **545**
33.1 frame help page man page . . . 545
33.2 Variables Used . . . 547
33.2.1 Primary variables . . . 547
33.2.2 Used variables . . . 548
33.3 Data Structures . . . 548
33.3.1 Frames and the Interpreter Frame Ring . . . 548
33.4 Accessor Functions . . . 548
33.4.1 0th Frame Component – frameName . . . 548
33.4.2 defun frameName . . . 548
33.4.3 1st Frame Component – frameInteractive . . . 549
33.4.4 2nd Frame Component – frameIOIndex . . . 549
33.4.5 3rd Frame Component – frameHiFiAccess . . . 549
33.4.6 4th Frame Component – frameHistList . . . 549
33.4.7 5th Frame Component – frameHistListLen . . . 550
33.4.8 6th Frame Component – frameHistListAct . . . 550
33.4.9 7th Frame Component – frameHistRecord . . . 550
33.4.10 8th Frame Component – frameHistoryTable . . . 550
33.4.11 9th Frame Component – frameExposureData . . . 551
33.5 Functions . . . 551
33.5.1 Initializing the Interpreter Frame Ring . . . 551
33.5.2 Creating a List of all of the Frame Names . . . 552
33.5.3 Get Named Frame Environment (aka Interactive) . . . 552
33.5.4 Create a new, empty Interpreter Frame . . . 552
33.5.5 Collecting up the Environment into a Frame . . . 553
33.5.6 Update from the Current Frame . . . 554
33.5.7 Find a Frame in the Frame Ring by Name . . . 555
33.5.8 Update the Current Interpreter Frame . . . 555
33.5.9 Move to the next Interpreter Frame in Ring . . . 556
33.5.10 Change to the Named Interpreter Frame . . . 556
33.5.11 Move to the previous Interpreter Frame in Ring . . . 557
33.5.12 Add a New Interpreter Frame . . . 557
33.5.13 Close an Interpreter Frame . . . 558
33.5.14 Display the Frame Names . . . 559
33.5.15 Import items from another frame . . . 559
33.5.16 The top level frame command . . . 561
33.5.17 The top level frame command handler . . . 562
33.6 Frame File Messages . . . 563

34)help help page Command **565**
34.1 help help page man page . . . 565
34.2 Functions . . . 568
34.2.1 The top level help command . . . 568
34.2.2 The top level help command handler . . . 568
34.2.3 defun newHelpSpad2Cmd . . . 568

35)history help page Command **571**
35.1 history help page man page . . . 571
35.2 Initialized history variables . . . 574
35.2.1 defvar $oldHistoryFileName . . . 574
35.2.2 defvar $historyFileType . . . 575
35.2.3 defvar $historyDirectory . . . 575
35.2.4 defvar $useInternalHistoryTable . . . 575
35.3 Data Structures . . . 575
35.4 Functions . . . 575
35.4.1 defun makeHistFileName . . . 575
35.4.2 defun oldHistFileName . . . 576
35.4.3 defun histFileName . . . 576
35.4.4 defun histInputFileName . . . 576
35.4.5 defun initHist . . . 577
35.4.6 defun initHistList . . . 577
35.4.7 The top level history command . . . 578
35.4.8 The top level history command handler . . . 578
35.4.9 defun setHistoryCore . . . 580
35.4.10 defvar $underbar . . . 582
35.4.11 defun writeInputLines . . . 583
35.4.12 defun resetInCoreHist . . . 584
35.4.13 defun changeHistListLen . . . 585
35.4.14 defun updateHist . . . 585
35.4.15 defun updateInCoreHist . . . 586
35.4.16 defun putHist . . . 586
35.4.17 defun recordNewValue . . . 587
35.4.18 defun recordNewValue0 . . . 587
35.4.19 defun recordOldValue . . . 588
35.4.20 defun recordOldValue0 . . . 588
35.4.21 defun undoInCore . . . 588
35.4.22 defun undoChanges . . . 589
35.4.23 defun undoFromFile . . . 590
35.4.24 defun saveHistory . . . 591
35.4.25 defun restoreHistory . . . 593
35.4.26 defun setIOindex . . . 595
35.4.27 defun showInput . . . 595
35.4.28 defun showInOut . . . 596
35.4.29 defun fetchOutput . . . 596
35.4.30 Read the history file using index n . . . 597

35.4.31 Write information of the current step to history file . . . 598
35.4.32 Disable history if an error occurred . . . 599
35.4.33 defun writeHistModesAndValues . . . 599
35.5 Lisplib output transformations . . . 600
35.5.1 defun spadrwrite0 . . . 600
35.5.2 defun Random write to a stream . . . 600
35.5.3 defun spadrwrite . . . 601
35.5.4 defun spadrread . . . 601
35.5.5 defun Random read a key from a stream . . . 601
35.5.6 defun unwritable? . . . 602
35.5.7 defun writifyComplain . . . 602
35.5.8 defun safeWritify . . . 602
35.5.9 defun writify,writifyInner . . . 603
35.5.10 defun writify . . . 606
35.5.11 defun spadClosure? . . . 607
35.5.12 defvar $NonNullStream . . . 607
35.5.13 defvar $NullStream . . . 607
35.5.14 defun dewritify,dewritifyInner . . . 608
35.5.15 defun dewritify . . . 611
35.5.16 defun ScanOrPairVec,ScanOrInner . . . 611
35.5.17 defun ScanOrPairVec . . . 612
35.5.18 defun gensymInt . . . 612
35.5.19 defun charDigitVal . . . 613
35.5.20 defun histFileErase . . . 613
35.6 History File Messages . . . 614

36)include help page Command 617
36.1 include help page man page . . . 617
36.2 Functions . . . 617
36.2.1 defun ncloopInclude1 . . . 617
36.2.2 Returns the first non-blank substring of the given string . . . 618
36.2.3 Open the include file and read it in . . . 618
36.2.4 Return the include filename . . . 618
36.2.5 Return the next token . . . 619

37)library help page Command 621
37.1 library help page man page . . . 621

38)lisp help page Command 623
38.1 lisp help page man page . . . 623
38.2 Functions . . . 624

39)load help page Command 625
39.1 load help page man page . . . 625
39.1.1 defun The)load command (obsolete) . . . 625

40)ltrace help page Command **627**
40.1 ltrace help page man page . . . 627
40.1.1 defun The top level)ltrace function . . . 628
40.2 Variables Used . . . 628
40.3 Functions . . . 628

41)pquit help page Command **629**
41.1 pquit help page man page . . . 629
41.2 Functions . . . 630
41.2.1 The top level pquit command . . . 630
41.2.2 The top level pquit command handler . . . 630

42)quit help page Command **633**
42.1 quit help page man page . . . 633
42.2 Functions . . . 634
42.2.1 The top level quit command . . . 634
42.2.2 The top level quit command handler . . . 634
42.2.3 Leave the Axiom interpreter . . . 635

43)read help page Command **637**
43.1 read help page man page . . . 637
43.1.1 defun The)read command . . . 638
43.1.2 defun Implement the)read command . . . 638
43.1.3 defun /read . . . 640

44)savesystem help page Command **641**
44.1 savesystem help page man page . . . 641
44.1.1 defun The)savesystem command . . . 642

45)set help page Command **643**
45.1 set help page man page . . . 643
45.2 Overview . . . 644
45.3 Variables Used . . . 645
45.4 Functions . . . 645
45.4.1 Initialize the set variables . . . 645
45.4.2 Reset the workspace variables . . . 646
45.4.3 Display the set option information . . . 647
45.4.4 Display the set variable settings . . . 649
45.4.5 Translate options values to t or nil . . . 650
45.4.6 Translate t or nil to option values . . . 651
45.5 The list structure . . . 651
45.6 breakmode . . . 652
45.6.1 defvar $BreakMode . . . 653
45.7 debug . . . 653
45.8 debug lambda type . . . 654
45.8.1 defvar $lambdatype . . . 654

45.9 debug dalymode . . . 654
45.9.1 defvar $dalymode . . . 655
45.10 compile . . . 655
45.11 compile output . . . 656
45.12 Variables Used . . . 656
45.13 Functions . . . 656
45.13.1 The set output command handler . . . 656
45.13.2 Describe the set output library arguments . . . 657
45.13.3 defvar $output-library . . . 657
45.13.4 Open the output library . . . 658
45.14 compile input . . . 658
45.15 Variables Used . . . 659
45.16 Functions . . . 659
45.16.1 The set input library command handler . . . 659
45.16.2 Describe the set input library arguments . . . 660
45.16.3 Add the input library to the list . . . 660
45.16.4 defvar $input-libraries . . . 660
45.16.5 Drop an input library from the list . . . 661
45.17 expose . . . 661
45.18 Variables Used . . . 662
45.18.1 defvar $globalExposureGroupAlist . . . 662
45.18.2 defvar $localExposureDataDefault . . . 688
45.18.3 defvar $localExposureData . . . 688
45.19 Functions . . . 688
45.19.1 The top level set expose command handler . . . 688
45.19.2 The top level set expose add command handler . . . 689
45.19.3 Expose a group . . . 690
45.19.4 The top level set expose add constructor handler . . . 692
45.19.5 The top level set expose drop handler . . . 693
45.19.6 The top level set expose drop group handler . . . 694
45.19.7 The top level set expose drop constructor handler . . . 695
45.19.8 Display exposed groups . . . 696
45.19.9 Display exposed constructors . . . 696
45.19.10 Display hidden constructors . . . 697
45.20 functions . . . 697
45.21 functions cache . . . 698
45.22 Variables Used . . . 699
45.22.1 defvar $cacheAlist . . . 699
45.23 Functions . . . 699
45.23.1 The top level set functions cache handler . . . 699
45.23.2 defvar $compileDontDefineFunctions . . . 703
45.24 functions recurrence . . . 703
45.24.1 defvar $compileRecurrence . . . 704
45.25 fortran . . . 704
45.25.1 ints2floats . . . 705
45.25.2 defvar $fortInts2Floats . . . 706

45.25.3 fortindent . . . 706
45.25.4 defvar $fortIndent . . . 706
45.25.5 fortlength . . . 707
45.25.6 defvar $fortLength . . . 707
45.25.7 typedecs . . . 707
45.25.8 defvar $printFortranDecs . . . 708
45.25.9 defaulttype . . . 708
45.25.10 defvar $defaultFortranType . . . 709
45.25.11 precision . . . 709
45.25.12 defvar $fortranPrecision . . . 709
45.25.13 intrinsic . . . 710
45.25.14 defvar $useIntrinsicFunctions . . . 710
45.25.15 explength . . . 711
45.25.16 defvar $maximumFortranExpressionLength . . . 711
45.25.17 segment . . . 711
45.25.18 defvar $fortranSegment . . . 712
45.25.19 optlevel . . . 712
45.25.20 defvar $fortranOptimizationLevel . . . 712
45.25.21 startindex . . . 713
45.25.22 defvar $fortranArrayStartingIndex . . . 713
45.25.23 calling . . . 713
45.25.24 defvar $fortranTmpDir . . . 714
45.25.25 The top level set fortran calling tempfile handler . . . 715
45.25.26 Validate the output directory . . . 716
45.25.27 Describe the set fortran calling tempfile . . . 716
45.25.28 defvar $fortranDirectory . . . 717
45.25.29 defun setFortDir . . . 717
45.25.30 defun describeSetFortDir . . . 718
45.25.31 defvar $fortranLibraries . . . 719
45.25.32 defun setLinkerArgs . . . 720
45.25.33 defun describeSetLinkerArgs . . . 720
45.26 hyperdoc . . . 721
45.26.1 fullscreen . . . 721
45.26.2 defvar $fullScreenSysVars . . . 721
45.26.3 mathwidth . . . 722
45.26.4 defvar $historyDisplayWidth . . . 722
45.27 help . . . 723
45.27.1 fullscreen . . . 723
45.27.2 defvar $useFullScreenHelp . . . 723
45.28 history . . . 724
45.28.1 defvar $HiFiAccess . . . 724
45.29 messages . . . 725
45.29.1 any . . . 726
45.29.2 defvar $printAnyIfTrue . . . 726
45.29.3 autoload . . . 727
45.29.4 defvar $printLoadMsgs . . . 727

45.29.5 bottomup . . . 728
45.29.6 defvar $reportBottomUpFlag . . . 728
45.29.7 coercion . . . 728
45.29.8 defvar $reportCoerceIfTrue . . . 729
45.29.9 dropmap . . . 729
45.29.10 defvar $displayDroppedMap . . . 730
45.29.11 expose . . . 730
45.29.12 defvar $giveExposureWarning . . . 730
45.29.13 file . . . 731
45.29.14 defvar $printMsgsToFile . . . 731
45.29.15 frame . . . 732
45.29.16 defvar $frameMessages . . . 732
45.29.17 highlighting . . . 732
45.29.18 defvar $highlightAllowed . . . 733
45.29.19 instant . . . 733
45.29.20 defvar $reportInstantiations . . . 734
45.29.21 insteach . . . 734
45.29.22 defvar $reportEachInstantiation— . . . 734
45.29.23 interponly . . . 735
45.29.24 defvar $reportInterpOnly . . . 735
45.29.25 naglink . . . 736
45.29.26 defvar $nagMessages . . . 736
45.29.27 number . . . 736
45.29.28 defvar $displayMsgNumber . . . 737
45.29.29 prompt . . . 737
45.29.30 defvar $inputPromptType . . . 738
45.29.31 selection . . . 738
45.29.32 set . . . 739
45.29.33 defvar $displaySetValue . . . 739
45.29.34 startup . . . 740
45.29.35 defvar $displayStartMsgs . . . 740
45.29.36 summary . . . 740
45.29.37 defvar $printStatisticsSummaryIfTrue . . . 741
45.29.38 testing . . . 741
45.29.39 defvar $testingSystem . . . 742
45.29.40 time . . . 742
45.29.41 defvar $printTimeIfTrue . . . 742
45.29.42 type . . . 743
45.29.43 defvar $printTypeIfTrue . . . 743
45.29.44 void . . . 744
45.29.45 defvar $printVoidIfTrue . . . 744
45.30 naglink . . . 744
45.30.1 host . . . 745
45.30.2 defvar $nagHost . . . 745
45.30.3 defun setNagHost . . . 746
45.30.4 defun describeSetNagHost . . . 746

45.30.5 persistence . . . 747
45.30.6 defvar $fortPersistence . . . 747
45.30.7 defun setFortPers . . . 748
45.30.8 defun describeFortPersistence . . . 748
45.30.9 messages . . . 749
45.30.10 double . . . 749
45.30.11 defvar $nagEnforceDouble . . . 750
45.31 output . . . 750
45.31.1 abbreviate . . . 751
45.31.2 defvar $abbreviateTypes . . . 752
45.31.3 algebra . . . 752
45.31.4 defvar $algebraFormat . . . 753
45.31.5 defvar $algebraOutputFile . . . 753
45.31.6 defvar $algebraOutputStream . . . 754
45.31.7 defun setOutputAlgebra . . . 754
45.31.8 defun describeSetOutputAlgebra . . . 756
45.31.9 characters . . . 757
45.31.10 defun setOutputCharacters . . . 758
45.31.11 fortran . . . 759
45.31.12 defvar $fortranFormat . . . 760
45.31.13 defvar $fortranOutputFile . . . 760
45.31.14 defun setOutputFortran . . . 761
45.31.15 defun describeSetOutputFortran . . . 763
45.31.16 fraction . . . 764
45.31.17 defvar $fractionDisplayType . . . 765
45.31.18 length . . . 765
45.31.19 defvar $margin . . . 765
45.31.20 defvar $linelength . . . 765
45.31.21 mathml . . . 766
45.31.22 defvar $mathmlFormat . . . 767
45.31.23 defvar $mathmlOutputFile . . . 767
45.31.24 defun setOutputMathml . . . 768
45.31.25 defun describeSetOutputMathml . . . 770
45.31.26 html . . . 771
45.31.27 defvar $htmlFormat . . . 771
45.31.28 defvar $htmlOutputFile . . . 771
45.31.29 defun setOutputHtml . . . 772
45.31.30 defun describeSetOutputHtml . . . 774
45.31.31 openmath . . . 775
45.31.32 defvar $openMathFormat . . . 776
45.31.33 defvar $openMathOutputFile . . . 776
45.31.34 defun setOutputOpenMath . . . 777
45.31.35 defun describeSetOutputOpenMath . . . 779
45.31.36 script . . . 780
45.31.37 defvar $formulaFormat . . . 781
45.31.38 defvar $formulaOutputFile . . . 781

45.31.39 defun setOutputFormula . . . 781
45.31.40 defun describeSetOutputFormula . . . 784
45.31.41 scripts . . . 784
45.31.42 defvar $linearFormatScripts . . . 785
45.31.43 showeditor . . . 785
45.31.44 defvar $useEditorForShowOutput . . . 786
45.31.45 tex . . . 786
45.31.46 defvar $texFormat . . . 787
45.31.47 defvar $texOutputFile . . . 787
45.31.48 defun setOutputTex . . . 788
45.31.49 defun describeSetOutputTex . . . 790
45.32 quit . . . 791
45.32.1 defvar $quitCommandType . . . 791
45.33 streams . . . 791
45.33.1 calculate . . . 792
45.33.2 defvar $streamCount . . . 792
45.33.3 defun setStreamsCalculate . . . 793
45.33.4 defun describeSetStreamsCalculate . . . 793
45.33.5 showall . . . 794
45.33.6 defvar $streamsShowAll . . . 794
45.34 system . . . 795
45.34.1 functioncode . . . 795
45.34.2 defvar $reportCompilation . . . 795
45.34.3 optimization . . . 796
45.34.4 defvar $reportOptimization . . . 796
45.34.5 prettyprint . . . 797
45.34.6 defvar $prettyprint . . . 797
45.35 userlevel . . . 798
45.35.1 defvar $UserLevel . . . 798
45.35.2 defvar $setOptionNames . . . 799
45.36 Set code . . . 799
45.36.1 defun set . . . 799
45.36.2 defun set1 . . . 800

46)show help page Command 805
46.1 show help page man page . . . 805
46.1.1 defun The)show command . . . 806
46.1.2 defun The internal)show command . . . 806
46.1.3 defun reportOperations . . . 807
46.1.4 defun reportOpsFromLisplib0 . . . 809
46.1.5 defun reportOpsFromLisplib1 . . . 809
46.1.6 defun reportOpsFromLisplib . . . 810
46.1.7 defun isExposedConstructor . . . 812
46.1.8 defun displayOperationsFromLisplib . . . 812
46.1.9 defun reportOpsFromUnitDirectly0 . . . 813
46.1.10 defun reportOpsFromUnitDirectly . . . 813

46.1.11 defun getOplistForConstructorForm . . . 816
46.1.12 defun getOplistWithUniqueSignatures . . . 817
46.1.13 defun reportOpsFromUnitDirectly1 . . . 817
46.1.14 defun sayShowWarning . . . 818

47)spool help page Command 819
47.1 spool help page man page . . . 819

48)summary help page Command 821
48.1 summary help page man page . . . 821
48.1.1 defun summary . . . 822

49)synonym help page Command 823
49.1 synonym help page man page . . . 823
49.1.1 defun The)synonym command . . . 824
49.1.2 defun The)synonym command implementation . . . 824
49.1.3 defun Return a sublist of applicable synonyms . . . 825
49.1.4 defun Get the system command from the input line . . . 825
49.1.5 defun Remove system keyword . . . 826
49.1.6 defun processSynonymLine . . . 827

50)system help page Command 829
50.1 system help page man page . . . 829

51)trace help page Command 831
51.1 trace help page man page . . . 831
51.1.1 The trace global variables . . . 835
51.1.2 defvar $traceNoisely . . . 836
51.1.3 defvar $reportSpadTrace . . . 836
51.1.4 defvar $optionAlist . . . 836
51.1.5 defvar $tracedMapSignatures . . . 836
51.1.6 defvar $traceOptionList . . . 836
51.1.7 defun trace . . . 837
51.1.8 defun traceSpad2Cmd . . . 837
51.1.9 defun trace1 . . . 838
51.1.10 defun getTraceOptions . . . 842
51.1.11 defun saveMapSig . . . 843
51.1.12 defun getMapSig . . . 843
51.1.13 defun getTraceOption,hn . . . 843
51.1.14 defun getTraceOption . . . 844
51.1.15 defun traceOptionError . . . 847
51.1.16 defun resetTimers . . . 848
51.1.17 defun resetSpacers . . . 848
51.1.18 defun resetCounters . . . 848
51.1.19 defun ptimers . . . 849
51.1.20 defun pspacers . . . 849

51.1.21 defun pcounters . . . 850
51.1.22 defun transOnlyOption . . . 850
51.1.23 defun stackTraceOptionError . . . 851
51.1.24 defun removeOption . . . 851
51.1.25 defun domainToGenvar . . . 851
51.1.26 defun genDomainTraceName . . . 852
51.1.27 defun untrace . . . 852
51.1.28 defun transTraceItem . . . 853
51.1.29 defun removeTracedMapSigs . . . 854
51.1.30 defun coerceTraceArgs2E . . . 854
51.1.31 defun coerceSpadArgs2E . . . 855
51.1.32 defun subTypes . . . 856
51.1.33 defun coerceTraceFunValue2E . . . 857
51.1.34 defun coerceSpadFunValue2E . . . 858
51.1.35 defun isListOfIdentifiers . . . 858
51.1.36 defun isListOfIdentifiersOrStrings . . . 859
51.1.37 defun getMapSubNames . . . 859
51.1.38 defun getPreviousMapSubNames . . . 860
51.1.39 defun lassocSub . . . 861
51.1.40 defun rassocSub . . . 861
51.1.41 defun isUncompiledMap . . . 861
51.1.42 defun isInterpOnlyMap . . . 862
51.1.43 defun augmentTraceNames . . . 862
51.1.44 defun isSubForRedundantMapName . . . 863
51.1.45 defun untraceMapSubNames . . . 863
51.1.46 defun funfind,LAM . . . 864
51.1.47 defmacro funfind . . . 864
51.1.48 defun isDomainOrPackage . . . 865
51.1.49 defun isTraceGensym . . . 865
51.1.50 defun spadTrace,g . . . 865
51.1.51 defun spadTrace,isTraceable . . . 865
51.1.52 defun spadTrace . . . 866
51.1.53 defun traceDomainLocalOps . . . 870
51.1.54 defun untraceDomainLocalOps . . . 870
51.1.55 defun traceDomainConstructor . . . 870
51.1.56 defun untraceDomainConstructor,keepTraced? . . . 872
51.1.57 defun untraceDomainConstructor . . . 873
51.1.58 defun flattenOperationAlist . . . 873
51.1.59 defun mapLetPrint . . . 874
51.1.60 defun letPrint . . . 875
51.1.61 defun Identifier beginning with a sharpsign-number? . . . 876
51.1.62 defun Identifier beginning with a sharpsign? . . . 876
51.1.63 defun isgenvar . . . 876
51.1.64 defun letPrint2 . . . 877
51.1.65 defun letPrint3 . . . 878
51.1.66 defun getAliasIfTracedMapParameter . . . 879

51.1.67 defun getBpiNameIfTracedMap . . . 880
51.1.68 defun hasPair . . . 881
51.1.69 defun shortenForPrinting . . . 881
51.1.70 defun spadTraceAlias . . . 881
51.1.71 defun getOption . . . 882
51.1.72 defun reportSpadTrace . . . 882
51.1.73 defun orderBySlotNumber . . . 883
51.1.74 defun /tracereply . . . 884
51.1.75 defun spadReply,printName . . . 884
51.1.76 defun spadReply . . . 885
51.1.77 defun spadUntrace . . . 885
51.1.78 defun remover . . . 887
51.1.79 defun prTraceNames,fn . . . 888
51.1.80 defun prTraceNames . . . 888
51.1.81 defvar $constructors . . . 889
51.1.82 defun traceReply . . . 889
51.1.83 defun addTraceItem . . . 892
51.1.84 defun ?t . . . 892
51.1.85 defun tracelet . . . 894
51.1.86 defun breaklet . . . 895
51.1.87 defun stupidIsSpadFunction . . . 896
51.1.88 defun break . . . 896
51.1.89 defun compileBoot . . . 897

52)undo help page Command 899
52.1 undo help page man page . . . 899
52.2 Evaluation . . . 900
52.2.1 defun evalDomain . . . 903
52.2.2 defun mkEvalable . . . 903
52.2.3 defun mkEvalableUnion . . . 905
52.2.4 defun mkEvalableRecord . . . 905
52.2.5 defun mkEvalableMapping . . . 905
52.2.6 defun evaluateType . . . 906
52.2.7 defun Eval args passed to a constructor . . . 907
52.2.8 defvar $noEvalTypeMsg . . . 909
52.2.9 defun throwEvalTypeMsg . . . 909
52.2.10 defun makeOrdinal . . . 910
52.2.11 defun evaluateSignature . . . 910
52.3 Data Structures . . . 910
52.4 Functions . . . 911
52.4.1 Initial Undo Variables . . . 911
52.4.2 defvar $undoFlag . . . 911
52.4.3 defvar $frameRecord . . . 911
52.4.4 defvar $previousBindings . . . 911
52.4.5 defvar $reportUndo . . . 912
52.4.6 defun undo . . . 912

52.4.7 defun recordFrame . . . 913
52.4.8 defun diffAlist . . . 914
52.4.9 defun reportUndo . . . 917
52.4.10 defun clearFrame . . . 919
52.4.11 Undo previous n commands . . . 919
52.4.12 defun undoSteps . . . 920
52.4.13 defun undoSingleStep . . . 921
52.4.14 defun undoLocalModemapHack . . . 923
52.4.15 Remove undo lines from history write . . . 923

53)what help page Command 927
53.1 what help page man page . . . 927
53.1.1 defvar $whatOptions . . . 929
53.1.2 defun what . . . 929
53.1.3 defun whatSpad2Cmd,fixpat . . . 929
53.1.4 defun whatSpad2Cmd . . . 930
53.1.5 defun Show keywords for)what command . . . 931
53.1.6 defun The)what commands implementation . . . 931
53.1.7 defun Find all names contained in a pattern . . . 932
53.1.8 defun Find function of names contained in pattern . . . 933
53.1.9 defun satisfiesRegularExpressions . . . 933
53.1.10 defun filterAndFormatConstructors . . . 934
53.1.11 defun whatConstructors . . . 935
53.1.12 Display all operation names containing the fragment . . . 935

54)with help page Command 937
54.1 with help page man page . . . 937
54.1.1 defun with . . . 937

55)workfiles help page Command 939
55.1 workfiles help page man page . . . 939
55.1.1 defun workfiles . . . 939
55.1.2 defun workfilesSpad2Cmd . . . 939

56)zsystemdevelopment help page Command 943
56.1 zsystemdevelopment help page man page . . . 943
56.1.1 defun zsystemdevelopment . . . 943
56.1.2 defun zsystemDevelopmentSpad2Cmd . . . 943
56.1.3 defun zsystemdevelopment1 . . . 944

57 Handlers for Special Forms 947
57.0.4 defun getAndEvalConstructorArgument . . . 948
57.0.5 defun replaceSharps . . . 948
57.0.6 defun isDomainValuedVariable . . . 949
57.0.7 defun evalCategory . . . 949

58 Handling input files **951**
58.0.8 defun Handle .axiom.input file . . . 951
58.0.9 defvar $boot-line-stack . . . 951
58.0.10 defvar $in-stream . . . 951
58.0.11 defvar $out-stream . . . 952
58.0.12 defvar $file-closed . . . 952
58.0.13 defvar $echo-meta . . . 952
58.0.14 defvar $noSubsumption . . . 952
58.0.15 defvar $envHashTable . . . 953
58.0.16 defun Dynamically add bindings to the environment . . . 953
58.0.17 defun Fetch a property list for a symbol from CategoryFrame . . . 954
58.0.18 defun Search for a binding in the environment list . . . 954
58.0.19 defun Search for a binding in the current environment . . . 954
58.0.20 defun searchTailEnv . . . 955

59 File Parsing **957**
59.0.21 defun Bind a variable in the interactive environment . . . 957
59.0.22 defvar $line-handler . . . 957
59.0.23 defvar $spad-errors . . . 957
59.0.24 defvar $xtokenreader . . . 958
59.0.25 defun Initialize the spad reader . . . 958
59.0.26 defun spad-syntax-error . . . 959
59.0.27 defun spad-long-error . . . 959
59.0.28 defun spad-short-error . . . 960
59.0.29 defun spad-error-loc . . . 960
59.0.30 defun iostat . . . 960
59.0.31 defun next-lines-show . . . 961
59.0.32 defun token-stack-show . . . 961
59.0.33 defun ioclear . . . 962
59.0.34 defun Set boot-line-stack to nil . . . 962

60 Handling output **965**
60.1 Special Character Tables . . . 965
60.1.1 defvar $defaultSpecialCharacters . . . 965
60.1.2 defvar $plainSpecialCharacters0 . . . 966
60.1.3 defvar $plainSpecialCharacters1 . . . 966
60.1.4 defvar $plainSpecialCharacters2 . . . 967
60.1.5 defvar $plainSpecialCharacters3 . . . 967
60.1.6 defvar $plainRTspecialCharacters . . . 968
60.1.7 defvar $RTspecialCharacters . . . 968
60.1.8 defvar $specialCharacters . . . 969
60.1.9 defvar $specialCharacterAlist . . . 969
60.1.10 defun Look up a special character code for a symbol . . . 970

61 Stream and File Handling **971**
61.0.11 defun make-instream . . . 971
61.0.12 defun make-outstream . . . 971
61.0.13 defun make-appendstream . . . 972
61.0.14 defun defiostream . . . 972
61.0.15 defun shut . . . 972
61.0.16 defun eofp . . . 973
61.0.17 defun makeStream . . . 973
61.0.18 defun Construct a new input file name . . . 973
61.0.19 defun getDirectoryList . . . 974
61.0.20 defun probeName . . . 974
61.0.21 defun makeFullNamestring . . . 975
61.0.22 defun Replace a file by erase and rename . . . 975

62 The Spad Server Mechanism **977**
62.0.23 defun openserver . . . 977

63 Axiom Build-time Functions **979**
63.0.24 defun spad-save . . . 979

64 Exposure Groups **981**

65 Databases **983**
65.1 Database structure . . . 983
65.1.1 kaf File Format . . . 983
65.1.2 Database Files . . . 984
65.1.3 defstruct $database . . . 986
65.1.4 defvar $*defaultdomain-list* . . . 987
65.1.5 defvar $*operation-hash* . . . 987
65.1.6 defvar $*hasCategory-hash* . . . 987
65.1.7 defvar $*miss* . . . 988
65.1.8 Database streams . . . 988
65.1.9 defvar $*compressvector* . . . 988
65.1.10 defvar $*compressVectorLength* . . . 988
65.1.11 defvar $*compress-stream* . . . 989
65.1.12 defvar $*compress-stream-stamp* . . . 989
65.1.13 defvar $*interp-stream* . . . 989
65.1.14 defvar $*interp-stream-stamp* . . . 989
65.1.15 defvar $*operation-stream* . . . 989
65.1.16 defvar $*operation-stream-stamp* . . . 990
65.1.17 defvar $*browse-stream* . . . 990
65.1.18 defvar $*browse-stream-stamp* . . . 990
65.1.19 defvar $*category-stream* . . . 990
65.1.20 defvar $*category-stream-stamp* . . . 991
65.1.21 defvar $*allconstructors* . . . 991
65.1.22 defvar $*allOperations* . . . 991

65.1.23 defun Reset all hash tables before saving system . . . 991
65.1.24 defun Preload algebra into saved system . . . 992
65.1.25 defun Open the interp database . . . 994
65.1.26 defun Open the browse database . . . 996
65.1.27 defun Open the category database . . . 997
65.1.28 defun Open the operations database . . . 998
65.1.29 defun Add operations from newly compiled code . . . 998
65.1.30 defun Show all database attributes of a constructor . . . 999
65.1.31 defun Set a value for a constructor key in the database . . . 1000
65.1.32 defun Delete a value for a constructor key in the database . . . 1001
65.1.33 defun Get constructor information for a database key . . . 1001
65.1.34 defun The `)library` top level command . . . 1005
65.1.35 defun Read a local filename and update the hash tables . . . 1005
65.1.36 defun Update the database from an nrlib index.kaf file . . . 1007
65.1.37 defun updateDatabase . . . 1009
65.1.38 defun Make new databases . . . 1009
65.1.39 defun saveDependentsHashTable . . . 1013
65.1.40 defun saveUsersHashTable . . . 1014
65.1.41 defun Construct the proper database full pathname . . . 1014
65.1.42 compress.daase . . . 1015
65.1.43 defun Set up compression vectors for the databases . . . 1015
65.1.44 defvar $*attributes* . . . 1016
65.1.45 defun Write out the compress database . . . 1016
65.1.46 defun Compress an expression using the compress vector . . . 1017
65.1.47 defun Uncompress an expression using the compress vector . . . 1018
65.1.48 Building the interp.daase from hash tables . . . 1018
65.1.49 defun Write the interp database . . . 1022
65.1.50 Building the browse.daase from hash tables . . . 1024
65.1.51 defun Write the browse database . . . 1024
65.1.52 Building the category.daase from hash tables . . . 1025
65.1.53 defun Write the category database . . . 1025
65.1.54 Building the operation.daase from hash tables . . . 1026
65.1.55 defun Write the operations database . . . 1026
65.1.56 Database support operations . . . 1027
65.1.57 defun Data preloaded into the image at build time . . . 1027
65.1.58 defun Return all constructors . . . 1027
65.1.59 defun Return all operations . . . 1027

66 System Statistics **1029**
66.1 Lisp Library Handling . . . 1029
66.1.1 defun loadLib . . . 1029
66.1.2 defun isSystemDirectory . . . 1030
66.1.3 defun loadLibNoUpdate . . . 1031
66.1.4 defun loadFunctor . . . 1032

67 Special Lisp Functions 1033
67.1 Axiom control structure macros . . . 1033
67.1.1 defun put . . . 1033
67.1.2 defmacro while . . . 1033
67.1.3 defmacro whileWithResult . . . 1034
67.2 Filename Handling . . . 1034
67.2.1 defun namestring . . . 1034
67.2.2 defun pathnameName . . . 1034
67.2.3 defun pathnameType . . . 1034
67.2.4 defun pathnameTypeId . . . 1035
67.2.5 defun mergePathnames . . . 1035
67.2.6 defun pathnameDirectory . . . 1035
67.2.7 defun Axiom pathnames . . . 1036
67.2.8 defun makePathname . . . 1036
67.2.9 defun Delete a file . . . 1036
67.2.10 defun wrap . . . 1037
67.2.11 defun lotsof . . . 1037
67.2.12 defmacro startsId? . . . 1038
67.2.13 defun hput . . . 1038
67.2.14 defmacro hget . . . 1038
67.2.15 defun hkeys . . . 1038
67.2.16 defun digitp . . . 1039
67.2.17 defun pname . . . 1039
67.2.18 defun size . . . 1039
67.2.19 defun strpos . . . 1039
67.2.20 defun strposl . . . 1040
67.2.21 defun qenum . . . 1040
67.2.22 defmacro identp . . . 1040
67.2.23 defun concat . . . 1041
67.2.24 defun functionp . . . 1041
67.2.25 defun brightprint . . . 1041
67.2.26 defun brightprint-0 . . . 1042
67.2.27 defun member . . . 1042
67.2.28 defun messageprint . . . 1042
67.2.29 defun messageprint-1 . . . 1043
67.2.30 defun messageprint-2 . . . 1043
67.2.31 defun sayBrightly1 . . . 1043
67.2.32 defmacro assq . . . 1044

68 Record, Union, Mapping, and Enumeration 1045

69 Common Lisp Algebra Support 1047
69.1 Void . . . 1047
69.1.1 defun voidValue . . . 1047
69.2 U32Vector . . . 1048
69.2.1 defun getrefv32 . . . 1048

69.2.2 defmacro qv32len . . . 1048
69.2.3 defmacro elt32 . . . 1048
69.2.4 defmacro setelt32 . . . 1048
69.3 DirectProduct . . . 1049
69.3.1 defun vec2list . . . 1049
69.4 AlgebraicFunction . . . 1049
69.4.1 defun retract . . . 1049
69.5 Any . . . 1051
69.5.1 defun spad2BootCoerce . . . 1051
69.6 ParametricLinearEquations . . . 1051
69.6.1 defun algCoerceInteractive . . . 1051
69.7 NumberFormats . . . 1052
69.7.1 defun ncParseFromString . . . 1052
69.8 SingleInteger . . . 1052
69.8.1 defun qsquotient . . . 1052
69.8.2 defun qsremainder . . . 1052
69.8.3 defmacro qsdifference . . . 1052
69.8.4 defmacro qslessp . . . 1053
69.8.5 defmacro qsadd1 . . . 1053
69.8.6 defmacro qssub1 . . . 1053
69.8.7 defmacro qsminus . . . 1053
69.8.8 defmacro qsplus . . . 1054
69.8.9 defmacro qstimes . . . 1054
69.8.10 defmacro qsabsval . . . 1054
69.8.11 defmacro qsoddp . . . 1054
69.8.12 defmacro qszerop . . . 1055
69.8.13 defmacro qsmax . . . 1055
69.8.14 defmacro qsmin . . . 1055
69.9 Boolean . . . 1055
69.9.1 defun The Boolean = function support . . . 1055
69.10 IndexedBits . . . 1056
69.10.1 defmacro truth-to-bit . . . 1056
69.10.2 defun IndexedBits new function support . . . 1056
69.10.3 defmacro bit-to-truth . . . 1056
69.10.4 defmacro bvec-elt . . . 1056
69.10.5 defmacro bvec-setelt . . . 1057
69.10.6 defmacro bvec-size . . . 1057
69.10.7 defun IndexedBits concat function support . . . 1057
69.10.8 defun IndexedBits copy function support . . . 1057
69.10.9 defun IndexedBits = function support . . . 1057
69.10.10 defun IndexedBits < function support . . . 1058
69.10.11 defun IndexedBits And function support . . . 1058
69.10.12 defun IndexedBits Or function support . . . 1058
69.10.13 defun IndexedBits xor function support . . . 1058
69.10.14 defun IndexedBits nand function support . . . 1059
69.10.15 defun IndexedBits nor function support . . . 1059

69.10.16defun IndexedBits not function support . . . 1059
69.11KeyedAccessFile . . . 1059
69.11.1 defun KeyedAccessFile defstream function support . . . 1059
69.11.2 defun KeyedAccessFile defstream function support . . . 1060
69.12Table . . . 1060
69.12.1 defun Table InnerTable support . . . 1060
69.12.2 defun compiledLookup . . . 1061
69.12.3 defun basicLookup . . . 1061
69.12.4 defun lookupInDomainVector . . . 1063
69.12.5 defun basicLookupCheckDefaults . . . 1063
69.12.6 defun oldCompLookup . . . 1064
69.12.7 defun NRTevalDomain . . . 1064
69.13Plot3d . . . 1065
69.13.1 defvar $numericFailure . . . 1065
69.13.2 defvar $oldBreakMode . . . 1065
69.13.3 defmacro trapNumericErrors . . . 1065
69.14DoubleFloatVector . . . 1066
69.14.1 defmacro dlen . . . 1066
69.14.2 defmacro make-double-vector . . . 1066
69.14.3 defmacro make-double-vector1 . . . 1066
69.14.4 defmacro delt . . . 1067
69.14.5 defmacro dsetelt . . . 1067
69.15ComplexDoubleFloatVector . . . 1067
69.15.1 defmacro make-cdouble-vector . . . 1067
69.15.2 defmacro cdelt . . . 1067
69.15.3 defmacro cdsetelt . . . 1068
69.15.4 defmacro cdlen . . . 1068
69.16DoubleFloatMatrix . . . 1069
69.16.1 defmacro make-double-matrix . . . 1069
69.16.2 defmacro make-double-matrix1 . . . 1069
69.16.3 defmacro daref2 . . . 1069
69.16.4 defmacro dsetaref2 . . . 1069
69.16.5 defmacro danrows . . . 1070
69.16.6 defmacro dancols . . . 1070
69.17ComplexDoubleFloatMatrix . . . 1070
69.17.1 defmacro make-cdouble-matrix . . . 1070
69.17.2 defmacro cdaref2 . . . 1070
69.17.3 defmacro cdsetaref2 . . . 1071
69.17.4 defmacro cdanrows . . . 1071
69.17.5 defmacro cdancols . . . 1072
69.18Integer . . . 1072
69.18.1 defun Integer divide function support . . . 1072
69.18.2 defun Integer quo function support . . . 1072
69.18.3 defun Integer quo function support . . . 1073
69.18.4 defun Integer random function support . . . 1073
69.19IndexCard . . . 1073

69.19.1 defun IndexCard origin function support . . . 1073
69.19.2 defun IndexCard origin function support . . . 1074
69.19.3 defun IndexCard elt function support . . . 1074
69.20 OperationsQuery . . . 1074
69.20.1 defun OperationQuery getDatabase function support . . . 1074
69.21 Database . . . 1075
69.21.1 defun Database elt function support . . . 1075
69.22 FileName . . . 1075
69.22.1 defun FileName filename function implementation . . . 1075
69.22.2 defun FileName filename support function . . . 1076
69.22.3 defun FileName directory function implementation . . . 1076
69.22.4 defun FileName directory function support . . . 1076
69.22.5 defun FileName name function implementation . . . 1077
69.22.6 defun FileName extension function implementation . . . 1077
69.22.7 defun FileName exists? function implementation . . . 1077
69.22.8 defun FileName readable? function implementation . . . 1077
69.22.9 defun FileName writeable? function implementation . . . 1078
69.22.10 defun FileName writeable? function support . . . 1078
69.22.11 defun FileName new function implementation . . . 1078
69.23 DoubleFloat . . . 1079
69.23.1 defmacro DFLessThan . . . 1079
69.23.2 defmacro DFUnaryMinus . . . 1079
69.23.3 defmacro DFMinusp . . . 1079
69.23.4 defmacro DFZerop . . . 1079
69.23.5 defmacro DFAdd . . . 1080
69.23.6 defmacro DFSubtract . . . 1080
69.23.7 defmacro DFMultiply . . . 1080
69.23.8 defmacro DFIntegerMultiply . . . 1080
69.23.9 defmacro DFMax . . . 1081
69.23.10 defmacro DFMin . . . 1081
69.23.11 defmacro DFEql . . . 1081
69.23.12 defmacro DFDivide . . . 1081
69.23.13 defmacro DFIntegerDivide . . . 1082
69.23.14 defmacro DFSqrt . . . 1082
69.23.15 defmacro DFLogE . . . 1082
69.23.16 defmacro DFLog . . . 1082
69.23.17 defmacro DFIntegerExpt . . . 1083
69.23.18 defmacro DFExpt . . . 1083
69.23.19 defmacro DFExp . . . 1083
69.23.20 defmacro DFSin . . . 1083
69.23.21 defmacro DFCos . . . 1084
69.23.22 defmacro DFTan . . . 1084
69.23.23 defmacro DFAsin . . . 1084
69.23.24 defmacro DFAcos . . . 1084
69.23.25 defmacro DFAtan . . . 1085
69.23.26 defmacro DFAtan2 . . . 1085

69.23.27 defmacro DFSinh . . . 1085
69.23.28 defmacro DFCosh . . . 1086
69.23.29 defmacro DFTanh . . . 1086
69.23.30 defmacro DFAsinh . . . 1086
69.23.31 defmacro DFAcosh . . . 1087
69.23.32 defmacro DFAtanh . . . 1087
69.23.33 defun Machine specific float numerator . . . 1087
69.23.34 defun Machine specific float denominator . . . 1088
69.23.35 defun Machine specific float sign . . . 1088
69.23.36 defun Machine specific float bit length . . . 1088
69.23.37 defun Decode floating-point values . . . 1088
69.23.38 defun The cotangent routine . . . 1089
69.23.39 defun The inverse cotangent function . . . 1089
69.23.40 defun The secant function . . . 1089
69.23.41 defun The inverse secant function . . . 1090
69.23.42 defun The cosecant function . . . 1090
69.23.43 defun The inverse cosecant function . . . 1090
69.23.44 defun The hyperbolic cosecant function . . . 1091
69.23.45 defun The hyperbolic cotangent function . . . 1091
69.23.46 defun The hyperbolic secant function . . . 1091
69.23.47 defun The inverse hyperbolic cosecant function . . . 1091
69.23.48 defun The inverse hyperbolic cotangent function . . . 1092
69.23.49 defun The inverse hyperbolic secant function . . . 1092

70 OpenMath 1093
70.1 A Technical Overview[4] . . . 1093
70.1.1 The OpenMath Architecture . . . 1093
70.1.2 OpenMath Encodings . . . 1095
70.1.3 Content Dictionaries . . . 1096
70.1.4 OpenMath in Action . . . 1098
70.2 Technical Details[3] . . . 1099
70.3 The Structure of the API . . . 1099
70.4 OpenMath Expressions . . . 1100
70.4.1 Expressions . . . 1100
70.4.2 Symbols . . . 1100
70.4.3 Encoding and Decoding OpenMath Expressions . . . 1100
70.5 Big Integers . . . 1101
70.6 Functions Dealing with OpenMath Devices . . . 1101
70.7 Functions to Write OpenMath Expressions to Devices . . . 1102
70.7.1 Beginning and Ending Objects . . . 1102
70.7.2 Writing Basic Objects . . . 1103
70.7.3 Writing Structured Objects . . . 1103
70.8 Functions to Extract OpenMath Expressions from Devices . . . 1104
70.8.1 Testing the type of the current token . . . 1104
70.8.2 Extracting the current token . . . 1105
70.9 Comments in the SGML/XML Encodings . . . 1108

70.10 I/O Functions for Devices . . . 1109
70.11 Communications . . . 1109
70.11.1 Functions to Initiate an OMconn . . . 1110
70.12 Parameters . . . 1111
70.13 Miscellaneous Functions and Variables . . . 1111
70.14 The OM.h header file . . . 1112
70.15 Axiom OpenMath stub functions . . . 1121
70.15.1 Axiom specific functions . . . 1121
70.15.2 defun om-Read . . . 1121
70.15.3 defun om-listCDs . . . 1122
70.15.4 defun om-listSymbols . . . 1122
70.15.5 defun om-supportsCD . . . 1122
70.15.6 defun om-supportsSymbol . . . 1122
70.15.7 Lisp conversion functions . . . 1123
70.15.8 defun om-setDevEncoding . . . 1123
70.15.9 Device manipulation functions . . . 1123
70.15.10 defun om-openFileDev . . . 1123
70.15.11 defun om-openStringDev . . . 1124
70.15.12 defun om-closeDev . . . 1124
70.15.13 Connection manipulation functions . . . 1124
70.15.14 defun om-makeConn . . . 1124
70.15.15 defun om-closeConn . . . 1124
70.15.16 defun om-getConnInDev . . . 1125
70.15.17 defun om-getConnOutDev . . . 1125
70.15.18 Client/Server functions . . . 1125
70.15.19 defun om-bindTCP . . . 1125
70.15.20 defun om-connectTCP . . . 1126
70.15.21 Device input/output functions . . . 1126
70.15.22 defun om-getApp . . . 1127
70.15.23 defun om-getAtp . . . 1128
70.15.24 defun om-getAttr . . . 1128
70.15.25 defun om-getBind . . . 1128
70.15.26 defun om-getBVar . . . 1128
70.15.27 defun om-getByteArray . . . 1129
70.15.28 defun om-getEndApp . . . 1129
70.15.29 defun om-getEndAtp . . . 1129
70.15.30 defun om-getEndAttr . . . 1129
70.15.31 defun om-getEndBind . . . 1130
70.15.32 defun om-getEndBVar . . . 1130
70.15.33 defun om-getEndError . . . 1130
70.15.34 defun om-getEndObject . . . 1130
70.15.35 defun om-getError . . . 1131
70.15.36 defun om-getFloat . . . 1131
70.15.37 defun om-getInt . . . 1131
70.15.38 defun om-getObject . . . 1131
70.15.39 defun om-getString . . . 1132

70.15.40 defun om-getSymbol . . . 1132
70.15.41 defun om-getType . . . 1132
70.15.42 defun om-getVar . . . 1132
70.15.43 defun om-putApp . . . 1133
70.15.44 defun om-putAtp . . . 1133
70.15.45 defun om-putAttr . . . 1133
70.15.46 defun om-putBind . . . 1133
70.15.47 defun om-putBVar . . . 1134
70.15.48 defun om-putByteArray . . . 1134
70.15.49 defun om-putEndApp . . . 1134
70.15.50 defun om-putEndAtp . . . 1134
70.15.51 defun om-putEndAttr . . . 1135
70.15.52 defun om-putEndBind . . . 1135
70.15.53 defun om-putEndBVar . . . 1135
70.15.54 defun om-putEndError . . . 1135
70.15.55 defun om-putEndObject . . . 1136
70.15.56 defun om-putError . . . 1136
70.15.57 defun om-putFloat . . . 1136
70.15.58 defun om-putInt . . . 1136
70.15.59 defun om-putObject . . . 1137
70.15.60 defun om-putString . . . 1137
70.15.61 defun om-putSymbol . . . 1137
70.15.62 defun om-putVar . . . 1137
70.15.63 defun om-stringToStringPtr . . . 1138
70.15.64 defun om-stringPtrToString . . . 1138

71 NRLIB code.lisp support code 1139
71.0.65 defun makeByteWordVec2 . . . 1139
71.0.66 defmacro spadConstant . . . 1139

72 Monitoring execution 1141
72.0.67 defvar $*monitor-domains* . . . 1147
72.0.68 defvar $*monitor-nrlibs* . . . 1147
72.0.69 defvar $*monitor-table* . . . 1148
72.0.70 defstruct $monitor-data . . . 1148
72.0.71 defstruct $libstream . . . 1148
72.0.72 defun Initialize the monitor statistics hashtable . . . 1148
72.0.73 defun End the monitoring process, we cannot restart . . . 1149
72.0.74 defun Return a list of the monitor-data structures . . . 1149
72.0.75 defun Add a function to be monitored . . . 1150
72.0.76 defun Remove a function being monitored . . . 1150
72.0.77 defun Enable all (or optionally one) function for monitoring . . . 1150
72.0.78 defun Disable all (optionally one) function for monitoring . . . 1151
72.0.79 defun Reset the table count for the table (or a function) . . . 1151
72.0.80 defun Incr the count of fn by 1 . . . 1152
72.0.81 defun Decr the count of fn by 1 . . . 1152

72.0.82 defun Return the monitor information for a function . . . 1153
72.0.83 defun Hang a monitor call on all of the defuns in a file . . . 1153
72.0.84 defun Return a list of the functions with zero count fields . . . 1153
72.0.85 defun Return a list of functions with non-zero counts . . . 1154
72.0.86 defun Write out a list of symbols or structures to a file . . . 1154
72.0.87 defun Save the *monitor-table* in loadable form . . . 1155
72.0.88 defun restore a checkpointed file . . . 1155
72.0.89 defun Printing help documentation . . . 1156
72.0.90 Monitoring algebra files . . . 1158
72.0.91 defun Monitoring algebra code.lsp files . . . 1158
72.0.92 defun Monitor autoloaded files . . . 1158
72.0.93 defun Monitor an nrlib . . . 1159
72.0.94 defun Given a monitor-data item, extract the nrlib name . . . 1159
72.0.95 defun Is this an exposed algebra function? . . . 1160
72.0.96 defun Monitor exposed domains . . . 1160
72.0.97 defun Generate a report of the monitored domains . . . 1161
72.0.98 defun Parse an)abbrev expression for the domain name . . . 1162
72.0.99 defun Given a spad file, report all nrlibs it creates . . . 1162
72.0.100 defun Print percent of functions tested . . . 1163
72.0.101 defun Find all monitored symbols containing the string . . . 1163

73 The Interpreter 1165

74 The Global Variables 1199
74.1 Star Global Variables . . . 1199
74.1.1 *eof* . . . 1199
74.1.2 *features* . . . 1199
74.1.3 *package* . . . 1199
74.1.4 *standard-input* . . . 1200
74.1.5 *standard-output* . . . 1200
74.1.6 *top-level-hook* . . . 1200
74.2 Dollar Global Variables . . . 1202
74.2.1 $boot . . . 1203
74.2.2 coerceFailure . . . 1203
74.2.3 $currentLine . . . 1203
74.2.4 $displayStartMsgs . . . 1203
74.2.5 $e . . . 1203
74.2.6 $erMsgToss . . . 1203
74.2.7 $fn . . . 1203
74.2.8 $frameRecord . . . 1203
74.2.9 $HiFiAccess . . . 1204
74.2.10 $HistList . . . 1204
74.2.11 $HistListAct . . . 1204
74.2.12 $HistListLen . . . 1204
74.2.13 $HistRecord . . . 1204
74.2.14 $historyFileType . . . 1205

74.2.15 $internalHistoryTable . . . 1205
74.2.16 $interpreterFrameName . . . 1205
74.2.17 $interpreterFrameRing . . . 1205
74.2.18 $InteractiveFrame . . . 1205
74.2.19 $intRestart . . . 1205
74.2.20 $intTopLevel . . . 1205
74.2.21 $IOindex . . . 1206
74.2.22 $lastPos . . . 1206
74.2.23 $libQuiet . . . 1206
74.2.24 $msgDatabaseName . . . 1206
74.2.25 $ncMsgList . . . 1206
74.2.26 $newcompErrorCount . . . 1206
74.2.27 $newspad . . . 1206
74.2.28 $nopos . . . 1206
74.2.29 $oldHistoryFileName . . . 1207
74.2.30 $okToExecuteMachineCode . . . 1207
74.2.31 $options . . . 1207
74.2.32 $previousBindings . . . 1207
74.2.33 $PrintCompilerMessageIfTrue . . . 1207
74.2.34 $reportUndo . . . 1207
74.2.35 $spad . . . 1207
74.2.36 $SpadServer . . . 1208
74.2.37 $SpadServerName . . . 1208
74.2.38 $systemCommandFunction . . . 1208
74.2.39 top_level . . . 1208
74.2.40 $quitTag . . . 1208
74.2.41 $useInternalHistoryTable . . . 1208
74.2.42 $undoFlag . . . 1208

75 Index **1211**

New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation's website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we've broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We've also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I'm looking forward to future milestones.

With that in mind I've introduced the theme of the "30 year horizon". We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How can we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The "30 year horizon" is much nearer than it appears.

Tim Daly
CAISS, City College of New York
November 10, 2003 ((iHy))

Chapter 1

Credits

Axiom has a very long history and many people have contributed to the effort, some in large ways and some in small ways. Any and all effort deserves recognition. There is no other criteria than contribution of effort. We would like to acknowledge and thank the following people:

1.0.1 defvar $credits

— **initvars** —

```
(defvar credits '(
"An alphabetical listing of contributors to AXIOM:"
"Michael Albaugh        Cyril Alberga          Roy Adler"
"Christian Aistleitner  Richard Anderson       George Andrews"
"S.J. Atkins"
"Henry Baker            Martin Baker           Stephen Balzac"
"Yurij Baransky         David R. Barton        Gerald Baumgartner"
"Gilbert Baumslag       Michael Becker         Nelson H. F. Beebe"
"Jay Belanger           David Bindel           Fred Blair"
"Vladimir Bondarenko    Mark Botch             Alexandre Bouyer"
"Peter A. Broadbery     Martin Brock           Manuel Bronstein"
"Stephen Buchwald       Florian Bundschuh      Luanne Burns"
"William Burge"
"Quentin Carpent        Robert Caviness        Bruce Char"
"Ondrej Certik          Cheekai Chin           David V. Chudnovsky"
"Gregory V. Chudnovsky  James Cloos            Josh Cohen"
"Christophe Conil       Don Coppersmith        George Corliss"
"Robert Corless         Gary Cornell           Meino Cramer"
"Claire Di Crescenzo    David Cyganski"
"Nathaniel Daly         Timothy Daly Sr.       Timothy Daly Jr."
"James H. Davenport     Didier Deshommes       Michael Dewar"
```

"Jean Della Dora" "Gabriel Dos Reis" "Claire DiCrescendo"
"Sam Dooley" "Lionel Ducos" "Lee Duhem"
"Martin Dunstan" "Brian Dupee" "Dominique Duval"
"Robert Edwards" "Heow Eide-Goodman" "Lars Erickson"
"Richard Fateman" "Bertfried Fauser" "Stuart Feldman"
"John Fletcher" "Brian Ford" "Albrecht Fortenbacher"
"George Frances" "Constantine Frangos" "Timothy Freeman"
"Korrinn Fu"
"Marc Gaetano" "Rudiger Gebauer" "Kathy Gerber"
"Patricia Gianni" "Samantha Goldrich" "Holger Gollan"
"Teresa Gomez-Diaz" "Laureano Gonzalez-Vega" "Stephen Gortler"
"Johannes Grabmeier" "Matt Grayson" "Klaus Ebbe Grue"
"James Griesmer" "Vladimir Grinberg" "Oswald Gschnitzer"
"Jocelyn Guidry"
"Gaetan Hache" "Steve Hague" "Satoshi Hamaguchi"
"Mike Hansen" "Richard Harke" "Bill Hart"
"Vilya Harvey" "Martin Hassner" "Arthur S. Hathaway"
"Dan Hatton" "Waldek Hebisch" "Karl Hegbloom"
"Ralf Hemmecke" "Henderson" "Antoine Hersen"
"Roger House" "Gernot Hueber"
"Pietro Iglio"
"Alejandro Jakubi" "Richard Jenks"
"Kai Kaminski" "Grant Keady" "Wilfrid Kendall"
"Tony Kennedy" "Ted Kosan" "Paul Kosinski"
"Klaus Kusche" "Bernhard Kutzler"
"Tim Lahey" "Larry Lambe" "Kaj Laurson"
"Franz Lehner" "Frederic Lehobey" "Michel Levaud"
"Howard Levy" "Liu Xiaojun" "Rudiger Loos"
"Michael Lucks" "Richard Luczak"
"Camm Maguire" "Francois Maltey" "Alasdair McAndrew"
"Bob McElrath" "Michael McGettrick" "Ian Meikle"
"David Mentre" "Victor S. Miller" "Gerard Milmeister"
"Mohammed Mobarak" "H. Michael Moeller" "Michael Monagan"
"Marc Moreno-Maza" "Scott Morrison" "Joel Moses"
"Mark Murray"
"William Naylor" "Patrice Naudin" "C. Andrew Neff"
"John Nelder" "Godfrey Nolan" "Arthur Norman"
"Jinzhong Niu"
"Michael O'Connor" "Summat Oemrawsingh" "Kostas Oikonomou"
"Humberto Ortiz-Zuazaga"
"Julian A. Padget" "Bill Page" "David Parnas"
"Susan Pelzel" "Michel Petitot" "Didier Pinchon"
"Ayal Pinkus" "Jose Alfredo Portes"
"Claude Quitte"
"Arthur C. Ralfs" "Norman Ramsey" "Anatoly Raportirenko"
"Albert D. Rich" "Michael Richardson" "Guilherme Reis"
"Renaud Rioboo" "Jean Rivlin" "Nicolas Robidoux"
"Simon Robinson" "Raymond Rogers" "Michael Rothstein"
"Martin Rubey"
"Philip Santas" "Alfred Scheerhorn" "William Schelter"

```
"Gerhard Schneider      Martin Schoenert       Marshall Schor"
"Frithjof Schulze       Fritz Schwarz          Steven Segletes"
"Nick Simicich          William Sit            Elena Smirnova"
"Jonathan Steinbach     Fabio Stumbo           Christine Sundaresan"
"Robert Sutor           Moss E. Sweedler       Eugene Surowitz"
"Max Tegmark            T. Doug Telford        James Thatcher"
"Balbir Thomas          Mike Thomas            Dylan Thurston"
"Steve Toleque          Barry Trager           Themos T. Tsikas"
"Gregory Vanuxem"
"Bernhard Wall          Stephen Watt           Jaap Weel"
"Juergen Weiss          M. Weller              Mark Wegman"
"James Wen              Thorsten Werther       Michael Wester"
"John M. Wiley          Berhard Will           Clifton J. Williamson"
"Stephen Wilson         Shmuel Winograd        Robert Wisbauer"
"Sandra Wityak          Waldemar Wiwianka      Knut Wolf"
"Clifford Yapp          David Yun"
"Vadim Zhytnikov        Richard Zippel         Evelyn Zoernack"
"Bruno Zuercher         Dan Zwillinger"
))
```

———-

Chapter 2

The Interpreter

The Axiom interpreter is a large common lisp program. It has several forms of interaction and run from terminal in a standalone fashion, run under the control of a session handler program, run as a web server, or run in a unix pipe.

Chapter 3

The Fundamental Data Structures

Axiom currently depends on a lot of global variables. These are generally listed here along with explanations.

3.1 The global variables

3.1.1 defvar $current-directory

The `$current-directory` variable is set to the current directory at startup. This is used by the `)cd` function and some of the compile routines. This is the result of the (p36) get-current-directory function. This variable is used to set `*default-pathname-defaults*`. The (p40) reroot function resets it to `$spadroot`.

An example of a runtime value is:

```
$current-directory = "/research/test/"
```

3.1.2 defvar $current-directory

— **initvars** —

```
(defvar $current-directory nil)
```

————-

3.1.3 defvar $defaultMsgDatabaseName

The `$defaultMsgDatabaseName` variable contains the location of the international message database. This can be changed to use a translated version of the messages. It defaults to the United States English version. The relative pathname used as the default is hardcoded in the (p40) reroot function. This value is prefixed with the `$spadroot` to make the path absolute.

In general, all Axiom message text should be stored in this file to enable internationalization of messages.

An example of a runtime value is:

```
|$defaultMsgDatabaseName| =
  #p"/research/test/mnt/ubuntu/doc/msgs/s2-us.msgs"
```

3.1.4 defvar $defaultMsgDatabaseName

— **initvars** —

```
(defvar |$defaultMsgDatabaseName| nil)
```

————-

3.1.5 defvar $directory-list

The `$directory-list` is a runtime list of absolute pathnames. This list is generated by the (p40) reroot function from the list of relative paths held in the variable `$relative-directory-list`. Each entry will be prefixed by `$spadroot`.

An example of a runtime value is:

```
$directory-list =
  ("/research/test/mnt/ubuntu/../../src/input/"
   "/research/test/mnt/ubuntu/doc/msgs/"
   "/research/test/mnt/ubuntu/../../src/algebra/"
   "/research/test/mnt/ubuntu/../../src/interp/"
   "/research/test/mnt/ubuntu/doc/spadhelp/")
```

3.1.6 defvar $directory-list

— **initvars** —

```
(defvar $directory-list nil)
```

———-

3.1.7 defvar $InitialModemapFrame

The `$InitialModemapFrame` is used as the initial value.

See the function "makeInitialModemapFrame" (5.3.14 p 37).

An example of a runtime value is:

```
$InitialModemapFrame = '((nil))
```

3.1.8 defvar $InitialModemapFrame

— **initvars** —

```
(defvar |$InitialModemapFrame| '((nil)))
```

———-

3.1.9 defvar $library-directory-list

The `$library-directory-list` variable is the system-wide search path for library files. It is set up in the (p40) reroot function by prepending the `$spadroot` variable to the `$relative-library-directory-list` variable.

An example of a runtime value is:

```
$library-directory-list =  ("/research/test/mnt/ubuntu/algebra/")
```

3.1.10 defvar $library-directory-list

— **initvars** —

```
(defvar $library-directory-list '("/algebra/"))
```

———-

3.1.11 defvar $msgDatabaseName

The `$msgDatabaseName` is a locally shared variable among the message database routines.

An example of a runtime value is:

```
|$msgDatabaseName| = nil
```

3.1.12 defvar $msgDatabaseName

— **initvars** —

```
(defvar |$msgDatabaseName| nil)
```

————-

3.1.13 defvar $openServerIfTrue

The `$openServerIfTrue` It appears to control whether the interpreter will be used as an open server, probably for OpenMath use.

If an open server is not requested then this variable to NIL

See the function "openserver" (62.0.23 p 977).

An example of a runtime value is:

```
$openServerIfTrue = nil
```

3.1.14 defvar $openServerIfTrue

— **initvars** —

```
(defvar $openServerIfTrue nil)
```

————-

3.1.15 defvar $relative-directory-list

The `$relative-directory-list` variable contains a hand-generated list of directories used in the Axiom system. The relative directory list specifies a search path for files for the current directory structure. It has been changed from the NAG distribution back to the original form.

This list is used by the (p40) reroot function to generate the absolute list of paths held in the variable `$directory-list`. Each entry will be prefixed by `$spadroot`.

An example of a runtime value is:

```
$relative-directory-list =
  ("/../../src/input/"
   "/doc/msgs/"
   "/../../src/algebra/"
   "/../../src/interp/"
   "/doc/spadhelp/")
```

3.1.16 defvar $relative-directory-list

— **initvars** —

```
(defvar $relative-directory-list
  '("/../../src/input/"
    "/doc/msgs/"
    "/../../src/algebra/"
    "/../../src/interp/"  ; for lisp  files (helps fd)
    "/doc/spadhelp/" ))
```

———-

3.1.17 defvar $relative-library-directory-list

The `$relative-library-directory-list` is a hand-generated list of directories containing algebra. The (p40) reroot function will prefix every path in this list with the value of the `$spadroot` variable to construct the `$library-directory-list` variable.

An example of a runtime value is:

```
$relative-library-directory-list = ("/algebra/")
```

3.1.18 defvar $relative-library-directory-list

— **initvars** —

```
(defvar $relative-library-directory-list '("/algebra/"))
```

———-

3.1.19 defvar $spadroot

The `$spadroot` variable is the internal name for the AXIOM shell variable. It is set in reroot to the value of the argument. The value is expected to be a directory name. The (p35)

initroot function uses this variable if the AXIOM shell variable is not set. The (p36) make-absolute-filename function uses this path as a prefix to all of the relative filenames to make them absolute.

An example of a runtime value is:

```
$spadroot = "/research/test/mnt/ubuntu"
```

3.1.20 defvar $spadroot

— **initvars** —

```
(defvar $spadroot nil)
```

————-

3.1.21 defvar $SpadServer

The `$SpadServer` determines whether Axiom acts as a remote server.

See the function "openserver" (62.0.23 p 977).

An example of a runtime value is:

```
$SpadServer = nil
```

3.1.22 defvar $SpadServer

— **initvars** —

```
(defvar $SpadServer nil "t means Axiom acts as a remote server")
```

————-

3.1.23 defvar $SpadServerName

The `$SpadServerName` defines the name of the spad server socket. In unix these exist in the tmp directory as names.

See the function "openserver" (62.0.23 p 977).

An example of a runtime value is:

```
$SpadServerName = "/tmp/.d"
```

3.1.24 defvar $SpadServerName

— **initvars** —

```
(defvar $SpadServerName "/tmp/.d" "the name of the spad server socket")
```

————-

Chapter 4

Starting Axiom

Axiom starts by invoking a function value of the lisp symbol `*top-level-hook*`. The function invocation path to from this point until the prompt is approximates (skipping initializations):

```
lisp -> restart
     -> |spad|
     -> |runspad|
     -> |ncTopLevel|
     -> |ncIntLoop|
     -> |intloop|
     -> |SpadInterpretStream|
     -> |intloopReadConsole|
```

The —intloopReadConsole— function does tail-recursive calls to itself (don't break this) and never exits.

4.1 Variables Used

4.2 Data Structures

4.3 Functions

4.3.1 Set the restart hook

When a lisp image containing code is reloaded there is a hook to allow a function to be called. In our case it is the restart function which is the entry to the Axiom interpreter.

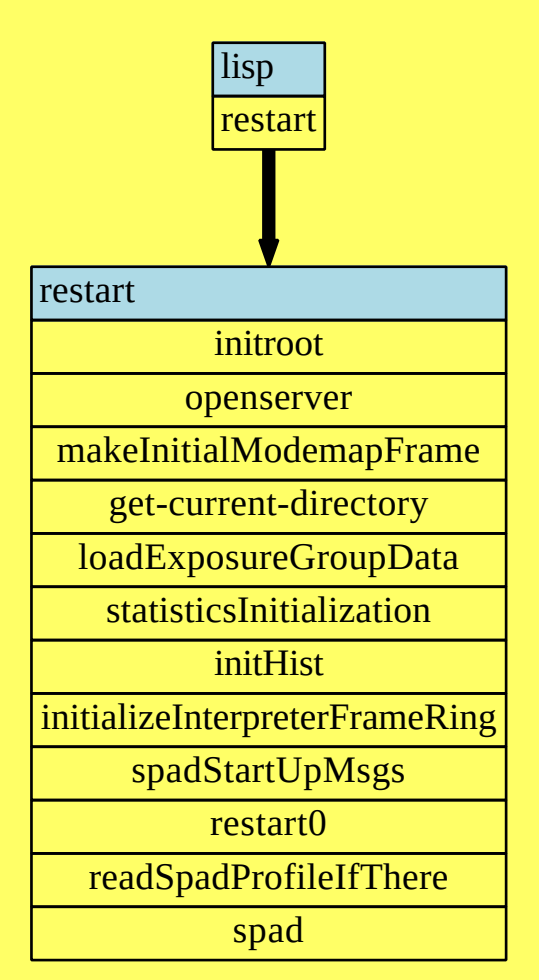
— **defun set-restart-hook 0** —

```
(defun set-restart-hook ()
```

```
  "Set the restart hook"
  #+KCL (setq system::*top-level-hook* 'restart)
  #+Lucid (setq boot::restart-hook 'restart)
  'restart
 )
```

———-

4.3.2 restart function (The restart function)



The restart function is the real root of the world. It sets up memory if we are working in a GCL/akcl version of the system.

The `compiler::*compile-verbose*` flag has been set to nil globally. We do not want to know

about the microsteps of GCL's compile facility.

The `compiler::*suppress-compiler-warnings*` flag has been set to t. We do not care that certain generated variables are not used.

The `compiler::*suppress-compiler-notes*` flag has been set to t. We do not care that tail recursion occurs.

It sets the current package to be the "BOOT" package which is the standard package in which the interpreter runs.

The "initroot" (5.3.9 p 35) function sets global variables that depend on the AXIOM shell variable. These are needed to find basic files like s2-us.msgs, which contains the error message text.

The "openserver" (62.0.23 p 977) function tried to set up the socket connection used for things like hyperdoc. The `$openServerIfTrue` variable starts true, which implies trying to start a server.

The `$IOindex` variable is the number associated with the input prompt. Every successful expression evaluated increments this number until a `)clear all` resets it. Here we set it to the initial value.

Axiom has multiple frames that contain independent information about a computation. There can be several frames at any one time and you can shift back and forth between the frames. By default, the system starts in "frame0" (try the `)frame names` command). See the Frame Mechanism chapter (33.3.1 page 548).

The `$InteractiveFrame` variable contains the state information related to the current frame, which includes things like the last value, the value of all of the variables, etc.

The "printLoadMsgs" (45.29.4 p 727) variable controls whether load messages will be output as library routines are loaded. We disnable this by default. It can be changed by using `)set message autoload`.

The "current-directory" (3.1.2 p 7) variable is set to the current directory. This is used by the `)cd` function and some of the compile routines.

The "statisticsInitialization" (?? p ??) function initializes variables used to collect statistics. Currently, only the garbage collector information is initialized.

[init-memory-config p34]
[initroot p35]
[openserver p977]
[makeInitialModemapFrame p37]
[get-current-directory p36]
[statisticsInitialization p??]
[initHist p577]
[initializeInterpreterFrameRing p551]
[spadStartUpMsgs p19]
[restart0 p18]
[readSpadProfileIfThere p951]
[spad p20]

[$openServerIfTrue p10]
[$SpadServername p??]
[$SpadServer p12]
[$IOindex p??]
[$InteractiveFrame p??]
[$printLoadMsgs p727]
[$current-directory p7]
[$displayStartMsgs p740]
[$currentLine p??]

— **defun restart** —

```
(defun restart ()
  (declare (special $openServerIfTrue $SpadServerName |$SpadServer|
    |$IOindex| |$InteractiveFrame| |$printLoadMsgs| $current-directory
    |$displayStartMsgs| |$currentLine|))
#+:akcl
  (init-memory-config :cons 500 :fixnum 200 :symbol 500 :package 8
    :array 400 :string 500 :cfun 100 :cpages 3000 :rpages 1000 :hole 2000)
#+:akcl (setq compiler::*compile-verbose* nil)
#+:akcl (setq compiler::*suppress-compiler-warnings* t)
#+:akcl (setq compiler::*suppress-compiler-notes* t)
  (in-package "BOOT")
  (initroot)
#+:akcl
  (when (and $openServerIfTrue (zerop (openserver $SpadServerName)))
    (setq $openServerIfTrue nil)
    (setq |$SpadServer| t))
  (setq |$IOindex| 1)
  (setq |$InteractiveFrame| (|makeInitialModemapFrame|))
  (setq |$printLoadMsgs| nil)
  (setq $current-directory (get-current-directory))
  (setq *default-pathname-defaults* (pathname $current-directory))
  (|statisticsInitialization|)
  (|initHist|)
  (|initializeInterpreterFrameRing|)
  (when |$displayStartMsgs| (|spadStartUpMsgs|))
  (setq |$currentLine| nil)
  (restart0)
  (|readSpadProfileIfThere|)
  (|spad|))
```

———-

4.3.3 defun Non-interactive restarts

[compressopen p??]
[interpopen p??]

[operationopen p??]
[categoryopen p??]
[browseopen p??]
[getEnv p??]

— **defun restart0** —

```
(defun restart0 ()
  (compressopen)    ;; set up the compression tables
  (interpopen)      ;; open up the interpreter database
  (operationopen)   ;; all of the operations known to the system
  (categoryopen)    ;; answer hasCategory question
  (browseopen))
```

———-

4.3.4 defun The startup banner messages

[fillerSpaces p20]
[specialChar p970]
[sayKeyedMsg p327]
[sayMSG p329]
[$msgAlist p324]
[$opSysName p??]
[$linelength p765]
[*yearweek* p??]
[*build-version* p??]

— **defun spadStartUpMsgs** —

```
(defun |spadStartUpMsgs| ()
 (let (bar)
 (declare (special |$msgAlist| |$opSysName| $linelength *yearweek*
                   *build-version*))
  (when (> $linelength 60)
   (setq bar (|fillerSpaces| $linelength (|specialChar| '|hbar|)))
   (|sayKeyedMsg| 'S2GL0001 (list *build-version* *yearweek*))
   (|sayMSG| bar)
   (|sayKeyedMsg| 'S2GL0018C nil)
   (|sayKeyedMsg| 'S2GL0018D nil)
   (|sayKeyedMsg| 'S2GL0003B (list |$opSysName|))
   (say "   Visit http://axiom-developer.org for more information")
   (|sayMSG| bar)
   (setq |$msgAlist| nil)
   (|sayMSG| '| |))))
```

———-

4.3.5 defun Make a vector of filler characters

[ifcar p??]

— **defun fillerSpaces** —

```
(defun |fillerSpaces| (&rest arglist &aux charPart n)
  (setq n (car arglist))
  (setq charPart (cdr arglist))
  (if (<= n 0)
   ""
   (make-string n :initial-element (character (or (ifcar charPart) " ")))))
```

———-

4.3.6 Starts the interpreter but do not read in profiles

[setOutputAlgebra p754]
[runspad p21]
[$PrintCompilerMessageIfTrue p??]

— **defun spad** —

```
(defun |spad| ()
 "Starts the interpreter but do not read in profiles"
 (let (|$PrintCompilerMessageIfTrue|)
  (declare (special |$PrintCompilerMessageIfTrue|))
  (setq |$PrintCompilerMessageIfTrue| nil)
  (|setOutputAlgebra| '|%initialize%|)
  (|runspad|)
  '|EndOfSpad|))
```

———-

4.3.7 defvar $quitTag

— **initvars** —

```
(defvar |$quitTag| system::*quit-tag*)
```

———-

4.3.8 defun runspad

[quitTag p20]
[coerceFailure p??]
[top-level p??]
[seq p??]
[exit p??]
[resetStackLimits p21]
[ncTopLevel p25]
[$quitTag p20]

— **defun runspad** —

```
(defun |runspad| ()
 (prog (mode)
 (declare (special |$quitTag|))
  (return
   (seq
    (progn
     (setq mode '|restart|)
     (do ()
         ((null (eq mode '|restart|)) nil)
      (seq
       (exit
        (progn
         (|resetStackLimits|)
         (catch |$quitTag|
          (catch '|coerceFailure|
           (setq mode (catch '|top_level| (|ncTopLevel|)))))))))))))
```

————-

4.3.9 defun Reset the stack limits

[reset-stack-limits p??]

— **defun resetStackLimits 0** —

```
(defun |resetStackLimits| ()
 "Reset the stack limits"
 (system:reset-stack-limits))
```

————-

Chapter 5

Handling Terminal Input

5.1 Streams

5.1.1 defvar $curinstream

The curinstream variable is set to the value of the `*standard-input*` common lisp variable in ncIntLoop. While not using the "dollar" convention this variable is still "global".

— **initvars** —

```
(defvar curinstream (make-synonym-stream '*standard-input*))
```

———-

5.1.2 defvar $curoutstream

The curoutstream variable is set to the value of the `*standard-output*` common lisp variable in ncIntLoop. While not using the "dollar" convention this variable is still "global".

— **initvars** —

```
(defvar curoutstream (make-synonym-stream '*standard-output*))
```

———-

5.1.3 defvar $errorinstream

— **initvars** —

```
(defvar errorinstream (make-synonym-stream '*terminal-io*))
```

———-

5.1.4 defvar $erroroutstream

— **initvars** —

```
(defvar erroroutstream (make-synonym-stream '*terminal-io*))
```

———-

5.1.5 defvar $*eof*

— **initvars** —

```
(defvar *eof* nil)
```

———-

5.1.6 defvar $*whitespace*

— **initvars** —

```
(defvar *whitespace*
 '(#\Space #\Newline #\Tab #\Page #\Linefeed #\Return #\Backspace)
 "A list of characters used by string-trim considered as whitespace")
```

———-

5.1.7 defvar $InteractiveMode

— **initvars** —

```
(defvar |$InteractiveMode| t)
```

———-

5.1.8 defvar $boot

— **initvars** —

```
(defvar $boot nil)
```

————-

5.1.9 Top-level read-parse-eval-print loop

Top-level read-parse-eval-print loop for the interpreter. Uses the Bill Burge's parser. [ncIntLoop p25]
[$e p??]
[$spad p20]
[$newspad p??]
[$boot p25]
[$InteractiveMode p24]
[$InteractiveFrame p??]
[*eof* p24]
[in-stream p951]

— **defun ncTopLevel** —

```
(defun |ncTopLevel| ()
 "Top-level read-parse-eval-print loop"
 (let (|$e| $spad $newspad $boot |$InteractiveMode| *eof* in-stream)
  (declare (special |$e| $spad $newspad $boot |$InteractiveMode| *eof*
            in-stream |$InteractiveFrame|))
   (setq in-stream curinstream)
   (setq *eof* nil)
   (setq |$InteractiveMode| t)
   (setq $boot nil)
   (setq $newspad t)
   (setq $spad t)
   (setq |$e| |$InteractiveFrame|)
   (|ncIntLoop|)))
```

————-

5.1.10 defun ncIntLoop

[intloop p26]
[curinstream p23]
[curoutstream p23]

— **defun ncIntLoop** —

```
(defun |ncIntLoop| ()
  (let ((curinstream *standard-output*)
        (curoutstream *standard-input*))
    (declare (special curinstream curoutstream))
    (|intloop|)))
```

————-

5.1.11 defvar $intTopLevel

— **initvars** —

```
(defvar |$intTopLevel| '|top_level|)
```

————-

5.1.12 defvar $intRestart

— **initvars** —

```
(defvar |$intRestart| '|restart|)
```

————-

5.1.13 defun intloop

Note that the SpadInterpretStream function uses a list of three strings as an argument. The values in the list seem to have no use and can eventually be removed. [intTopLevel p26]
[SpadInterpretStream p27]
[resetStackLimits p21]
[$intTopLevel p26]
[$intRestart p26]

— **defun intloop** —

```
(defun |intloop| ()
 (prog (mode)
  (declare (special |$intTopLevel| |$intRestart|))
```

```
(return
 (progn
  (setq mode |$intRestart|)
  ((lambda ()
    (loop
     (cond
      ((not (equal mode |$intRestart|))
        (return nil))
      (t
        (progn
          (|resetStackLimits|)
          (setq mode
            (catch |$intTopLevel|
              (|SpadInterpretStream| 1
                (list 'tim  'daly '?) t))))))))))))
```

———-

5.1.14 defvar $ncMsgList

— **initvars** —

```
(defvar |$ncMsgList| nil)
```

———-

5.1.15 defun SpadInterpretStream

The SpadInterpretStream function takes three arguments

str This is passed as an argument to intloopReadConsole

source This is the name of a source file but appears not to be used. It is set to the list `(tim daly ?)`.

`interactive?` If this is false then various messages are suppressed and input does not use piles. If this is true then the library loading routines might output messages and piles are expected on input (as from a file).

The system commands are handled by the function kept in the "hook" variable `$systemCommandFunction` which has the default function `InterpExecuteSpadSystemCommand`. Thus, when a system command is entered this function is called.

The `$promptMsg` variable is set to the constant S2CTP023. This constant points to a message in src/doc/msgs/s2-us.msgs. This message does nothing but print the argument value.

5.1.16 defvar $promptMsg

— **initvars** —

```
(defvar |$promptMsg| 'S2CTP023)
```

———-

5.1.17 defun GCL cmpnote function

GCL keeps noting the fact that the compiler is performing tail-recursion. Bill Schelter added this as a debugging tool for Axiom and it was never removed. Patching the lisp code in the GCL build fails as the system is actually built from the pre-compiled C code. Thus, we can only step on this message after the fact. The cmpnote function is used nowhere else in GCL so stepping on the function call seems best. We're unhappy with this hack and will try to convince the GCL crowd to fix this.

— **defun cmpnote** —

```
#+:gcl (defun compiler::cmpnote (&rest x) (declare (ignore x)))
```

———-

5.1.18 defvar $newcompErrorCount

— **initvars** —

```
(defvar |$newcompErrorCount| 0)
```

———-

5.1.19 defvar $nopos

— **initvars** —

```
(defvar |$nopos| (list '|noposition|))
```

———-

[mkprompt p42]
[intloopReadConsole p30]
[intloopInclude p63]
[$promptMsg p28]
[$systemCommandFunction p??]
[$ncMsgList p27]
[$erMsgToss p??]
[$lastPos p??]
[$inclAssertions p??]
[$okToExecuteMachineCode p??]
[$newcompErrorCount p28]
[$libQuiet p??]
[$fn p??]
[$nopos p28]

— **defun SpadInterpretStream** —

```
(defun |SpadInterpretStream| (str source interactive?)
 (let (|$promptMsg| |$systemCommandFunction|
       |$ncMsgList| |$erMsgToss| |$lastPos| |$inclAssertions|
       |$okToExecuteMachineCode| |$newcompErrorCount|
       |$libQuiet| |$fn|)
  (declare (special |$promptMsg|
            |$systemCommandFunction| |$ncMsgList| |$erMsgToss| |$lastPos|
            |$inclAssertions| |$okToExecuteMachineCode| |$newcompErrorCount|
            |$libQuiet| |$fn| |$nopos|))
  (setq |$fn| source)
  (setq |$libQuiet| (null interactive?))
  (setq |$newcompErrorCount| 0)
  (setq |$okToExecuteMachineCode| t)
  (setq |$inclAssertions| (list 'aix '|CommonLisp|))
  (setq |$lastPos| |$nopos|)
  (setq |$erMsgToss| nil)
  (setq |$ncMsgList| nil)
  (setq |$systemCommandFunction| #'|InterpExecuteSpadSystemCommand|)
  (setq |$promptMsg| 's2ctp023)
  (if interactive?
   (progn
     (princ (mkprompt))
     (|intloopReadConsole| "" str))
   (|intloopInclude| source 0))))
```

———-

5.2 The Read-Eval-Print Loop

5.2.1 defun intloopReadConsole

Note that this function relies on the fact that lisp can do tail-recursion. The function recursively invokes itself.

The serverReadLine function is a special readline function that handles communication with the session manager code, which is a separate process running in parallel.

We read a line from standard input.

- If it is a null line then we exit Axiom.
- If it is a zero length line we prompt and recurse
- If $dalymode and open-paren we execute lisp code, prompt and recurse The $dalymode will interpret any input that begins with an open-paren as a lisp expression rather than Axiom input. This is useful for debugging purposes when most of the input lines will be lisp. Setting $dalymode non-nil will certainly break user expectations and is to be used with caution.
- If it is ")fi" or ")fin" we drop into lisp. Use the (restart) function to return to the interpreter loop.
- If it starts with ")" we process the command, prompt, and recurse
- If it is a command then we remember the current line, process the command, prompt, and recurse.
- If the input has a trailing underscore (Axiom line-continuation) then we cut off the continuation character and pass the truncated string to ourselves, prompt, and recurse
- otherwise we process the input, prompt, and recurse.

Notice that all but two paths (a null input or a ")fi" or a ")fin") will end up as a recursive call to ourselves. [top-level p**??**]
[serverReadLine p44]
[leaveScratchpad p635]
[mkprompt p42]
[intloopReadConsole p30]
[intloopPrefix? p36]
[intnplisp p36]
[setCurrentLine p41]
[ncloopCommand p474]
[concat p1041]
[ncloopEscaped p37]
[intloopProcessString p37]
[$dalymode p655]

— **defun intloopReadConsole** —

```
(defun |intloopReadConsole| (b n)
 (declare (special $dalymode))
 (let (c d pfx input)
   (setq input (|serverReadLine| *standard-input*))
   (when (null (stringp input)) (|leaveScratchpad|))
   (when (eql (length input) 0)
     (princ (mkprompt))
     (|intloopReadConsole| "" n))
   (when (and $dalymode (|intloopPrefix?| "(" input))
     (|intnplisp| input)
     (princ (mkprompt))
     (|intloopReadConsole| "" n))
   (setq pfx (|intloopPrefix?| ")fi" input))
   (when (and pfx (or (string= pfx ")fi") (string= pfx ")fin")))
     (throw '|top_level| nil))
   (when (and (equal b "") (setq d (|intloopPrefix?| ")" input)))
     (|setCurrentLine| d)
     (setq c (|ncloopCommand| d n))
     (princ (mkprompt))
     (|intloopReadConsole| "" c))
   (setq input (concat b input))
   (when (|ncloopEscaped| input)
     (|intloopReadConsole| (subseq input 0 (- (length input) 1)) n))
   (setq c (|intloopProcessString| input n))
   (princ (mkprompt))
   (|intloopReadConsole| "" c)))
```

————-

5.3 Helper Functions

5.3.1 Get the value of an evironment variable

[getenv p**??**]

— **defun getenviron 0** —

```
(defun getenviron (var)
 "Get the value of an evironment variable"
 #+allegro (sys::getenv (string var))
 #+clisp (ext:getenv (string var))
 #+(or cmu scl)
  (cdr
   (assoc (string var) ext:*environment-list* :test #'equalp :key #'string))
```

```
#+(or kcl akcl gcl) (si::getenv (string var))
#+lispworks (lw:environment-variable (string var))
#+lucid (lcl:environment-variable (string var))
#+mcl (ccl::getenv var)
#+sbcl (sb-ext:posix-getenv var)
)
```

———-

5.3.2 defvar $intCoerceFailure

— **initvars** —

```
(defvar |$intCoerceFailure| '|coerceFailure|)
```

———-

5.3.3 defvar $intSpadReader

— **initvars** —

```
(defvar |$intSpadReader| 'SPAD_READER)
```

———-

5.3.4 defun InterpExecuteSpadSystemCommand

[intCoerceFailure p32]
[intSpadReader p32]
[ExecuteInterpSystemCommand p33]
[$intSpadReader p32]
[$intCoerceFailure p32]

— **defun InterpExecuteSpadSystemCommand** —

```
(defun |InterpExecuteSpadSystemCommand| (string)
 (declare (special |$intSpadReader| |$intCoerceFailure|))
  (catch |$intCoerceFailure|
   (catch |$intSpadReader|
    (|ExecuteInterpSystemCommand| string))))
```

———-

5.3.5 defun ExecuteInterpSystemCommand

[intProcessSynonyms p33]
[substring p??]
[doSystemCommand p442]
[$currentLine p??]

— **defun ExecuteInterpSystemCommand** —

```
(defun |ExecuteInterpSystemCommand| (string)
 (let (|$currentLine|)
 (declare (special |$currentLine|))
  (setq string (|intProcessSynonyms| string))
  (setq |$currentLine| string)
  (setq string (substring string 1 nil))
  (unless (equal string "") (|doSystemCommand| string))))
```

———-

5.3.6 defun Handle Synonyms

[processSynonyms p33]
[line p??]

— **defun intProcessSynonyms** —

```
(defun |intProcessSynonyms| (str)
 (let ((line str))
 (declare (special line))
  (|processSynonyms|)
  line))
```

———-

5.3.7 defun Synonym File Reader

[strpos p1039]
[substring p??]
[string2id-n p??]
[lassoc p??]
[strconc p??]
[size p1039]
[concat p1041]
[rplacstr p??]
[processSynonyms p33]
[$CommandSynonymAlist p474]

[line p**??**]

— **defun processSynonyms** —

```
(defun |processSynonyms| ()
 (let (fill p aline synstr syn to opt fun cl chr)
 (declare (special |$CommandSynonymAlist| line))
  (setq p (strpos ")" line 0 nil))
  (setq fill "")
  (cond
   (p
    (setq aline (substring line p nil))
    (when (> p 0) (setq fill (substring line 0 p))))
   (t
    (setq p 0)
    (setq aline line)))
  (setq to (strpos " " aline 1 nil))
  (cond (to (setq to (1- to))))
  (setq synstr (substring aline 1 to))
  (setq syn (string2id-n synstr 1))
  (when  (setq fun (lassoc syn |$CommandSynonymAlist|))
   (setq to (strpos ")" fun 1 nil))
   (cond
    ((and to (not (eql to (1- (size fun)))))
     (setq opt (strconc " " (substring fun to nil)))
     (setq fun (substring fun 0 (1- to ))))
    (t (setq opt " ")))
   (when (> (size synstr) (size fun))
    (do ((G167173 (size synstr)) (i (size fun) (1+ i)))
        ((> i G167173) nil)
     (setq fun (concat fun " "))))
   (setq cl (strconc fill (rplacstr aline 1 (size synstr) fun) opt))
   (setq line cl)
   (setq chr (elt line (1+ p)))
   (|processSynonyms|))))
```

————-

5.3.8 defun init-memory-config

Austin-Kyoto Common Lisp (AKCL), now known as Gnu Common Lisp (GCL) requires some changes to the default memory setup to run Axiom efficently. This function performs those setup commands. [allocate p**??**]
[allocate-contiguous-pages p**??**]
[allocate-relocatable-pages p**??**]
[set-hole-size p**??**]

— **defun init-memory-config 0** —

```
(defun init-memory-config (&key
                           (cons 500)
                           (fixnum 200)
                           (symbol 500)
                           (package 8)
                           (array 400)
                           (string 500)
                           (cfun 100)
                           (cpages 3000)
                           (rpages 1000)
                           (hole 2000) )
  ;; initialize AKCL memory allocation parameters
  #+:AKCL
  (progn
    (system:allocate 'cons cons)
    (system:allocate 'fixnum fixnum)
    (system:allocate 'symbol symbol)
    (system:allocate 'package package)
    (system:allocate 'array array)
    (system:allocate 'string string)
    (system:allocate 'cfun cfun)
    (system:allocate-contiguous-pages cpages)
    (system:allocate-relocatable-pages rpages)
    (system:set-hole-size hole))
  #-:AKCL
  nil)
```

———-

5.3.9 Set spadroot to be the AXIOM shell variable

Sets up the system to use the **AXIOM** shell variable if we can and default to the **$spadroot** variable (which was the value of the **AXIOM** shell variable at build time) if we can't.

[reroot p40]
[getenviron p31]
[$spadroot p12]

— **defun initroot** —

```
(defun initroot (&optional (newroot (getenviron "AXIOM")))
 "Set spadroot to be the AXIOM shell variable"
  (declare (special $spadroot))
  (reroot (or newroot $spadroot (error "setenv AXIOM or (setq $spadroot)"))))
```

———-

5.3.10 Does the string start with this prefix?

If the prefix string is the same as the whole string initial characters –R(ignoring spaces in the whole string) then we return the whole string minus any leading spaces.

— **defun intloopPrefix? 0** —

```
(defun |intloopPrefix?| (prefix whole)
 "Does the string start with this prefix?"
 (let ((newprefix (string-left-trim '(#\space) prefix))
       (newwhole  (string-left-trim '(#\space) whole)))
  (when (<= (length newprefix) (length newwhole))
   (when (string= newprefix newwhole :end2 (length prefix))
    newwhole))))
```

————-

5.3.11 defun Interpret a line of lisp code

This is used to hande `)lisp` top level commands [nplisp p468]
[$currentLine p**??**]

— **defun intnplisp** —

```
(defun |intnplisp| (s)
 (declare (special |$currentLine|))
 (setq |$currentLine| s)
 (|nplisp| |$currentLine|))
```

————-

5.3.12 Get the current directory

— **defun get-current-directory 0** —

```
(defun get-current-directory ()
 "Get the current directory"
  (namestring (truename "")))
```

————-

5.3.13 Prepend the absolute path to a filename

Prefix a filename with the **AXIOM** shell variable. [$spadroot p12]

— **defun make-absolute-filename 0** —

```
(defun make-absolute-filename (name)
 "Prepend the absolute path to a filename"
 (declare (special $spadroot))
 (concatenate 'string $spadroot name))
```

———-

5.3.14 Make the initial modemap frame

[copy p**??**]
[$InitialModemapFrame p9]

— **defun makeInitialModemapFrame 0** —

```
(defun |makeInitialModemapFrame| ()
  "Make the initial modemap frame"
  (declare (special |$InitialModemapFrame|))
  (copy |$InitialModemapFrame|))
```

———-

5.3.15 defun ncloopEscaped

The ncloopEscaped function will return true if the last non-blank character of a line is an underscore, the Axiom line-continuation character. Otherwise, it returns nil.

— **defun ncloopEscaped 0** —

```
(defun |ncloopEscaped| (x)
 (let ((l (length x)))
  (dotimes (i l)
   (when (char= (char x (- l i 1)) #\_) (return t))
   (unless (char= (char x (- l i 1)) #\space) (return nil)))))
```

———-

5.3.16 defun intloopProcessString

[setCurrentLine p41]
[intloopProcess p64]
[next p38]
[incString p39]

— **defun intloopProcessString** —

```
(defun |intloopProcessString| (s n)
 (|setCurrentLine| s)
 (|intloopProcess| n t
  (|next| #'|ncloopParse|
   (|next| #'|lineoftoks| (|incString| s)))))
```

———-

5.3.17 defun ncloopParse

[ncloopDQlines p71]
[npParse p139]
[dqToList p342]

— **defun ncloopParse** —

```
(defun |ncloopParse| (s)
 (let (cudr lines stream dq t1)
  (setq t1 (car s))
  (setq dq (car t1))
  (setq stream (cadr t1))
  (setq t1 (|ncloopDQlines| dq stream))
  (setq lines (car t1))
  (setq cudr (cadr t1))
  (cons (list (list lines (|npParse| (|dqToList| dq)))) (cdr s))))
```

———-

5.3.18 defun next

[Delay p103]
[next1 p38]

— **defun next** —

```
(defun |next| (f s)
 (|Delay| #'|next1| (list f s)))
```

———-

5.3.19 defun next1

[StreamNull p331]
[incAppend p87]

[next p38]

— **defun next1** —

```
(defun |next1| (&rest z)
 (let (h s f)
  (setq f (car z))
  (setq s (cadr z))
  (cond
   ((|StreamNull| s) |StreamNil|)
   (t
    (setq h (apply f (list s)))
    (|incAppend| (car h) (|next| f (cdr h)))))))
```

———-

5.3.20 defun incString

[incRenumber p74]
[incLude p76]
[Top p77]

— **defun incString** —

```
(defun |incString| (s)
 (declare (special |Top|))
 (|incRenumber| (|incLude| 0 (list s) 0 (list "strings") (list |Top|))))
```

———-

5.3.21 Call the garbage collector

Call the garbage collector on various platforms.

— **defun reclaim 0** —

```
#+abcl
(defun reclaim () "Call the garbage collector" (ext::gc))
#+:allegro
(defun reclaim () "Call the garbage collector" (excl::gc t))
#+:CCL
(defun reclaim () "Call the garbage collector" (gc))
#+clisp
(defun reclaim ()
 "Call the garbage collector"
 (#+lisp=cl ext::gc #-lisp=cl lisp::gc))
```

```
#+(or :cmulisp :cmu)
(defun reclaim () "Call the garbage collector" (ext:gc))
#+cormanlisp
(defun reclaim () "Call the garbage collector" (cl::gc))
#+(OR IBCL KCL GCL)
(defun reclaim () "Call the garbage collector" (si::gbc t))
#+lispworks
(defun reclaim () "Call the garbage collector" (hcl::normal-gc))
#+Lucid
(defun reclaim () "Call the garbage collector" (lcl::gc))
#+sbcl
(defun reclaim () "Call the garbage collector" (sb-ext::gc))
```

———-

5.3.22 defun reroot

The reroot function is used to reset the important variables used by the system. In particular, these variables are sensitive to the **AXIOM** shell variable. That variable is renamed internally to be **$spadroot**. The **reroot** function will change the system to use a new root directory and will have the same effect as changing the **AXIOM** shell variable and rerunning the system from scratch. Note that we have changed from the NAG distribution back to the original form. If you need the NAG version you can push **:tpd** on the ***features*** variable before compiling this file. A correct call looks like:

```
(in-package "BOOT")
(reroot "/spad/mnt/${SYS}")
```

where the `${SYS}` variable is the same one set at build time.

For the example call:

```
(REROOT "/research/test/mnt/ubuntu")
```

the variables are set as:

```
$spadroot = "/research/test/mnt/ubuntu"

$relative-directory-list =
  ("/../../src/input/"
   "/doc/msgs/"
   "/../../src/algebra/"
   "/../../src/interp/"
   "/doc/spadhelp/")

$directory-list =
  ("/research/test/mnt/ubuntu/../../src/input/"
```

```
   "/research/test/mnt/ubuntu/doc/msgs/"
   "/research/test/mnt/ubuntu/../../src/algebra/"
   "/research/test/mnt/ubuntu/../../src/interp/"
   "/research/test/mnt/ubuntu/doc/spadhelp/")

$relative-library-directory-list = ("/algebra/")

$library-directory-list =  ("/research/test/mnt/ubuntu/algebra/")

|$defaultMsgDatabaseName| = #p"/research/test/mnt/ubuntu/doc/msgs/s2-us.msgs"

|$msgDatabaseName| = nil

$current-directory = "/research/test/"
```

[make-absolute-filename p36]
[$spadroot p12]
[$directory-list p8]
[$relative-directory-list p11]
[$library-directory-list p9]
[$relative-library-directory-list p11]
[$defaultMsgDatabaseName p8]
[$msgDatabaseName p324]
[$current-directory p7]

— **defun reroot** —

```
(defun reroot (dir)
  (declare (special $spadroot $directory-list $relative-directory-list
     $library-directory-list $relative-library-directory-list
     |$defaultMsgDatabaseName| |$msgDatabaseName| $current-directory))
  (setq $spadroot dir)
  (setq $directory-list
   (mapcar #'make-absolute-filename $relative-directory-list))
  (setq $library-directory-list
   (mapcar #'make-absolute-filename $relative-library-directory-list))
  (setq |$defaultMsgDatabaseName|
        (pathname (make-absolute-filename "/doc/msgs/s2-us.msgs")))
  (setq |$msgDatabaseName| ())
  (setq $current-directory $spadroot))
```

————-

5.3.23 defun setCurrentLine

Remember the current line. The cases are:

- If there is no $currentLine set it to the input
- Is the current line a string and the input a string? Make them into a list
- Is $currentLine not a cons cell? Make it one.
- Is the input a string? Cons it on the end of the list.
- Otherwise stick it on the end of the list

Note I suspect the last two cases do not occur in practice since they result in a dotted pair if the input is not a cons. However, this is what the current code does so I won't change it. [$currentLine p**??**]

— **defun setCurrentLine 0** —

```
(defun |setCurrentLine| (s)
 (declare (special |$currentLine|))
 (cond
  ((null |$currentLine|) (setq |$currentLine| s))
  ((and (stringp |$currentLine|) (stringp s))
    (setq |$currentLine| (list |$currentLine| s)))
  ((not (consp |$currentLine|)) (setq |$currentLine| (cons |$currentLine| s)))
  ((stringp s) (rplacd (last |$currentLine|) (cons s nil)))
  (t (rplacd (last |$currentLine|) s)))
 |$currentLine|)
```

————-

5.3.24 Show the Axiom prompt

[concat p1041]
[substring p**??**]
[currenttime p**??**]
[$inputPromptType p738]
[$IOindex p**??**]
[$interpreterFrameName p**??**]

— **defun mkprompt** —

```
(defun mkprompt ()
 "Show the Axiom prompt"
 (declare (special |$inputPromptType| |$IOindex| |$interpreterFrameName|))
 (case |$inputPromptType|
  (|none| "")
  (|plain| "-> ")
  (|step|  (concat "(" (princ-to-string |$IOindex|) ") -> "))
```

```
  (|frame|
   (concat (princ-to-string |$interpreterFrameName|) " ("
           (princ-to-string |$IOindex|) ") -> "))
  (t (concat (princ-to-string |$interpreterFrameName|) " ["
             (substring (currenttime) 8 nil) "] ["
             (princ-to-string |$IOindex|) "] -> ")))
```

———-

5.3.25 defvar $frameAlist

— **initvars** —

```
(defvar |$frameAlist| nil)
```

———-

5.3.26 defvar $frameNumber

— **initvars** —

```
(defvar |$frameNumber| 0)
```

———-

5.3.27 defvar $currentFrameNum

— **initvars** —

```
(defvar |$currentFrameNum| 0)
```

———-

5.3.28 defvar $EndServerSession

— **initvars** —

```
(defvar |$EndServerSession| nil)
```

———-

5.3.29 defvar $NeedToSignalSessionManager

— **initvars** —

```
(defvar |$NeedToSignalSessionManager| nil)
```

———-

5.3.30 defvar $sockBufferLength

— **initvars** —

```
(defvar |$sockBufferLength| 9217)
```

———-

5.3.31 READ-LINE in an Axiom server system

[coerceFailure p**??**]
[top-level p**??**]
[spad-reader p**??**]
[read-line p**??**]
[addNewInterpreterFrame p557]
[sockSendInt p**??**]
[sockSendString p**??**]
[mkprompt p42]
[sockGetInt p**??**]
[lassoc p**??**]
[changeToNamedInterpreterFrame p556]
[sockGetString p**??**]
[unescapeStringsInForm p62]
[protectedEVAL p47]
[executeQuietCommand p47]
[parseAndInterpret p48]
[serverReadLine is-console (vol9)]

[serverSwitch p??]
[$KillLispSystem p??]
[$NonSmanSession p??]
[$SpadCommand p??]
[$QuietSpadCommand p??]
[$MenuServer p??]
[$sockBufferLength p44]
[$LispCommand p??]
[$EndServerSession p43]
[$EndSession p??]
[$SwitchFrames p??]
[$CreateFrameAnswer p??]
[$currentFrameNum p43]
[$frameNumber p43]
[$frameAlist p43]
[$CreateFrame p??]
[$CallInterp p??]
[$EndOfOutput p??]
[$SessionManager p??]
[$NeedToSignalSessionManager p44]
[$EndServerSession p43]
[$SpadServer p12]
[*eof* p24]
[in-stream p951]

— defun serverReadLine —

```
(defun |serverReadLine| (stream)
 "used in place of READ-LINE in a Axiom server system."
 (let (in-stream *eof* l framename currentframe form stringbuf line action)
 (declare (special in-stream *eof* |$SpadServer| |$EndServerSession|
     |$NeedToSignalSessionManager| |$SessionManager| |$EndOfOutput|
     |$CallInterp| |$CreateFrame| |$frameAlist| |$frameNumber|
     |$currentFrameNum| |$CreateFrameAnswer| |$SwitchFrames| |$EndSession|
     |$EndServerSession| |$LispCommand| |$sockBufferLength| |$MenuServer|
     |$QuietSpadCommand| |$SpadCommand| |$NonSmanSession| |$KillLispSystem|))
  (force-output)
  (if (or (null |$SpadServer|) (null (is-console stream)))
   (|read-line| stream)
   (progn
    (setq in-stream stream)
    (setq *eof* nil)
    (setq line
     (do ()
         ((null (and (null |$EndServerSession|) (null *eof*))) nil)
         (when |$NeedToSignalSessionManager|
          (|sockSendInt| |$SessionManager| |$EndOfOutput|))
      (setq |$NeedToSignalSessionManager| nil)
```

```
  (setq action (|serverSwitch|))
  (cond
   ((= action |$CallInterp|)
    (setq l (|read-line| stream))
    (setq |$NeedToSignalSessionManager| t)
    (return l))
   ((= action |$CreateFrame|)
    (setq framename (gentemp "frame"))
    (|addNewInterpreterFrame| framename)
    (setq |$frameAlist|
     (cons (cons |$frameNumber| framename) |$frameAlist|))
    (setq |$currentFrameNum| |$frameNumber|)
    (|sockSendInt| |$SessionManager| |$CreateFrameAnswer|)
    (|sockSendInt| |$SessionManager| |$frameNumber|)
    (setq |$frameNumber| (1+ |$frameNumber|))
    (|sockSendString| |$SessionManager| (mkprompt)))
   ((= action |$SwitchFrames|)
    (setq |$currentFrameNum| (|sockGetInt| |$SessionManager|))
    (setq currentframe (lassoc |$currentFrameNum| |$frameAlist|))
    (|changeToNamedInterpreterFrame| currentframe))
   ((= action |$EndSession|)
    (setq |$EndServerSession| t))
   ((= action |$LispCommand|)
    (setq |$NeedToSignalSessionManager| t)
    (setq stringbuf (make-string |$sockBufferLength|))
    (|sockGetString| |$MenuServer| stringbuf |$sockBufferLength|)
    (setq form (|unescapeStringsInForm| (read-from-string stringbuf)))
    (|protectedEVAL| form))
   ((= action |$QuietSpadCommand|)
    (setq |$NeedToSignalSessionManager| t)
    (|executeQuietCommand|))
   ((= action |$SpadCommand|)
    (setq |$NeedToSignalSessionManager| t)
    (setq stringbuf (make-string 512))
    (|sockGetString| |$MenuServer| stringbuf 512)
    (catch '|coerceFailure|
     (catch '|top_level|
      (catch 'spad_reader
       (|parseAndInterpret| stringbuf))))
    (princ (mkprompt))
    (finish-output))
   ((= action |$NonSmanSession|) (setq |$SpadServer| nil))
   ((= action |$KillLispSystem|) (bye))
   (t nil))))
(cond
 (line line)
 (t '|||||||))
```

5.3.32 defun protectedEVAL

[resetStackLimits p21]
[sendHTErrorSignal p??]

— **defun protectedEVAL** —

```
(defun |protectedEVAL| (x)
 (let (val (error t))
  (unwind-protect
   (progn
    (setq val (eval x))
    (setq error nil))
   (when error
    (|resetStackLimits|)
    (|sendHTErrorSignal|)))
  (unless error val)))
```

———-

5.3.33 defvar $QuietCommand

— **initvars** —

```
(defvar |$QuietCommand| nil "If true, produce no top level output")
```

———-

5.3.34 defun executeQuietCommand

When `$QuiteCommand` is true Spad will not produce any output from a top level command
[spad-reader p??]
[coerceFailure p??]
[toplevel p??]
[spadreader p??]
[make-string p??]
[sockGetString p??]
[parseAndInterpret p48]
[$MenuServer p??]
[$QuietCommand p47]

— **defun executeQuietCommand** —

```
(defun |executeQuietCommand| ()
 (let (|$QuietCommand| stringBuf)
 (declare (special |$QuietCommand| |$MenuServer|))
  (setq |$QuietCommand| t)
  (setq stringBuf (make-string 512))
  (|sockGetString| |$MenuServer| stringBuf 512)
  (catch '|coerceFailure|
   (catch '|top_level|
    (catch 'spad_reader (|parseAndInterpret| stringBuf))))))
```

———-

5.3.35 defun parseAndInterpret

[$InteractiveMode p24]
[$boot p25]
[$spad p20]
[$e p??]
[$InteractiveFrame p??]

— **defun parseAndInterpret** —

```
(defun |parseAndInterpret| (str)
 (let (|$InteractiveMode| $boot $spad |$e|)
 (declare (special |$InteractiveMode| $boot $spad |$e|
                   |$InteractiveFrame|))
  (setq |$InteractiveMode| t)
  (setq $boot nil)
  (setq $spad t)
  (setq |$e| |$InteractiveFrame|)
  (|processInteractive| (|parseFromString| str) nil)))
```

———-

5.3.36 defun parseFromString

[next p38]
[ncloopParse p38]
[lineoftoks p108]
[incString p39]
[StreamNull p331]
[pf2Sex p297]
[macroExpanded p220]

— **defun parseFromString** —

```
(defun |parseFromString| (s)
  (setq s (|next| #'|ncloopParse| (|next| #'|lineoftoks| (|incString| s))))
  (unless (|StreamNull| s) (|pf2Sex| (|macroExpanded| (cadar s)))))
```

———-

5.3.37 defvar $interpOnly

— **initvars** —

```
(defvar |$interpOnly| nil)
```

———-

5.3.38 defvar $minivectorNames

— **initvars** —

```
(defvar |$minivectorNames| nil)
```

———-

5.3.39 defvar $domPvar

— **initvars** —

```
(defvar |$domPvar| nil)
```

———-

5.3.40 defun processInteractive

Parser Output --> Interpreter

Top-level dispatcher for the interpreter. It sets local variables and then calls processInteractive1 to do most of the work. This function receives the output from the parser. [initializeTimedNames p**??**]
[qcar p**??**]

[processInteractive1 p52]
[reportInstantiations p734]
[clrhash p??]
[writeHistModesAndValues p599]
[updateHist p585]
[$op p??]
[$Coerce p??]
[$compErrorMessageStack p??]
[$freeVars p??]
[$mapList p??]
[$compilingMap p??]
[$compilingLoop p??]
[$interpOnly p49]
[$whereCacheList p??]
[$timeGlobalName p??]
[$StreamFrame p??]
[$declaredMode p??]
[$localVars p??]
[$analyzingMapList p??]
[$lastLineInSEQ p??]
[$instantCoerceCount p??]
[$instantCanCoerceCount p??]
[$instantMmCondCount p??]
[$fortVar p??]
[$minivector p??]
[$minivectorCode p??]
[$minivectorNames p49]
[$domPvar p49]
[$inRetract p??]
[$instantRecord p??]
[$reportInstantiations p734]
[$ProcessInteractiveValue p52]
[$defaultFortVar p??]
[$interpreterTimedNames p??]
[$interpreterTimedClasses p??]

— defun processInteractive —

```
(defun |processInteractive| (form posnForm)
 (let (|$op| |$Coerce| |$compErrorMessageStack| |$freeVars|
             |$mapList| |$compilingMap| |$compilingLoop|
             |$interpOnly| |$whereCacheList| |$timeGlobalName|
             |$StreamFrame| |$declaredMode| |$localVars|
             |$analyzingMapList| |$lastLineInSEQ|
             |$instantCoerceCount| |$instantCanCoerceCount|
             |$instantMmCondCount| |$fortVar| |$minivector|
             |$minivectorCode| |$minivectorNames| |$domPvar|
```

```
            |$inRetract| object)
 (declare (special |$op| |$Coerce| |$compErrorMessageStack|
                   |$freeVars| |$mapList| |$compilingMap|
                   |$compilingLoop| |$interpOnly| |$whereCacheList|
                   |$timeGlobalName| |$StreamFrame| |$declaredMode|
                   |$localVars| |$analyzingMapList| |$lastLineInSEQ|
                   |$instantCoerceCount| |$instantCanCoerceCount|
                   |$instantMmCondCount| |$fortVar| |$minivector|
                   |$minivectorCode| |$minivectorNames| |$domPvar|
                   |$inRetract| |$instantRecord| |$reportInstantiations|
                   |$ProcessInteractiveValue| |$defaultFortVar|
                   |$interpreterTimedNames| |$interpreterTimedClasses|))
  (|initializeTimedNames| |$interpreterTimedNames| |$interpreterTimedClasses|)
  (if (consp form)                        ; compute name of operator
   (setq |$op| (qcar form))
   (setq |$op| form))
  (setq |$Coerce| nil)
  (setq |$compErrorMessageStack| nil)
  (setq |$freeVars| nil)
  (setq |$mapList| nil)                   ; list of maps being type analyzed
  (setq |$compilingMap| nil)              ; true when compiling a map
  (setq |$compilingLoop| nil)             ; true when compiling a loop body
  (setq |$interpOnly| nil)                ; true when in interp only mode
  (setq |$whereCacheList| nil)            ; maps compiled because of where
  (setq |$timeGlobalName| '|$compTimeSum|); see incrementTimeSum
  (setq |$StreamFrame| nil)               ; used in printing streams
  (setq |$declaredMode| nil)              ; weak type propagation for symbols
  (setq |$localVars| nil)                 ; list of local variables in function
  (setq |$analyzingMapList| nil)       ; names of maps currently being analyzed
  (setq |$lastLineInSEQ| t)               ; see evalIF and friends
  (setq |$instantCoerceCount| 0)
  (setq |$instantCanCoerceCount| 0)
  (setq |$instantMmCondCount| 0)
  (setq |$defaultFortVar| 'x)             ; default FORTRAN variable name
  (setq |$fortVar| |$defaultFortVar|)     ; variable name for FORTRAN output
  (setq |$minivector| nil)
  (setq |$minivectorCode| nil)
  (setq |$minivectorNames| nil)
  (setq |$domPvar| nil)
  (setq |$inRetract| nil)
  (setq object (|processInteractive1| form posnForm))
  (unless |$ProcessInteractiveValue|
   (when |$reportInstantiations|
    (|reportInstantiations|)
    (clrhash |$instantRecord|))
   (|writeHistModesAndValues|)
   (|updateHist|))
  object))
```

———-

5.3.41 defvar $ProcessInteractiveValue

— **initvars** —

```
(defvar |$ProcessInteractiveValue| nil "If true, no output or record")
```

———-

5.3.42 defvar $HTCompanionWindowID

— **initvars** —

```
(defvar |$HTCompanionWindowID| nil)
```

———-

5.3.43 defun processInteractive1

This calls the analysis and output printing routines [recordFrame p913]
[startTimingProcess p**??**]
[interpretTopLevel p53]
[stopTimingProcess p**??**]
[recordAndPrint p56]
[objValUnwrap p**??**]
[objMode p**??**]
[$e p**??**]
[$ProcessInteractiveValue p52]
[$InteractiveFrame p**??**]

— **defun processInteractive1** —

```
(defun |processInteractive1| (form posnForm)
 (let (|$e| object)
 (declare (special |$e| |$ProcessInteractiveValue| |$InteractiveFrame|))
  (setq |$e| |$InteractiveFrame|)
  (|recordFrame| '|system|)
  (|startTimingProcess| '|analysis|)
  (setq object (|interpretTopLevel| form posnForm))
  (|stopTimingProcess| '|analysis|)
```

```
  (|startTimingProcess| '|print|)
  (unless |$ProcessInteractiveValue|
    (|recordAndPrint| (|objValUnwrap| object) (|objMode| object)))
  (|recordFrame| '|normal|)
  (|stopTimingProcess| '|print|)
  object))
```

———-

5.3.44 defun interpretTopLevel

[interpreter p??]
[interpret p54]
[stopTimingProcess p??]
[peekTimedName p??]
[interpretTopLevel p53]
[$timedNameStack p??]

— **defun interpretTopLevel** —

```
(defun |interpretTopLevel| (x posnForm)
 (let (savedTimerStack c)
 (declare (special |$timedNameStack|))
  (setq savedTimerStack (copy |$timedNameStack|))
  (setq c (catch '|interpreter| (|interpret| x posnForm)))
  (do ()
      ((equal savedTimerStack |$timedNameStack|) nil)
    (|stopTimingProcess| (|peekTimedName|)))
  (if (eq c '|tryAgain|)
   (|interpretTopLevel| x posnForm)
   c)))
```

———-

5.3.45 defvar $genValue

If the `$genValue` variable is true then evaluate generated code, otherwise leave code unevaluated. If `$genValue` is false then we are compiling. This variable is only defined and used locally.

— **initvars** —

```
(defvar |$genValue| nil "evaluate generated code if true")
```

———-

5.3.46 defun Type analyzes and evaluates expression x, returns object

[interpret1 p54]
[$env p??]
[$eval p??]
[$genValue p53]

— **defun interpret** —

```
(defun |interpret| (&rest arg &aux restargs x)
 (let (|$env| |$eval| |$genValue| posnForm)
 (declare (special |$env| |$eval| |$genValue|))
  (setq x (car arg))
  (setq restargs (cdr arg))
  (if (consp restargs)
    (setq posnForm (car restargs))
    (setq posnForm restargs))
  (setq |$env| (list (list nil)))
  (setq |$eval| t)      ; generate code -- don't just type analyze
  (setq |$genValue| t)  ; evaluate all generated code
  (|interpret1| x nil posnForm)))
```

———-

5.3.47 defun Dispatcher for the type analysis routines

This is the dispatcher for the type analysis routines. It type analyzes and evaluates the expression x in the rootMode (if non-nil) which may be `$EmptyMode`. It returns an object if evaluating, and a modeset otherwise. It creates the attributed tree. [mkAtreeWithSrcPos p??]
[putTarget p??]
[bottomUp p??]
[getArgValue p??]
[objNew p??]
[getValue p??]
[interpret2 p55]
[keyedSystemError p??]
[$genValue p53]
[$eval p??]

— **defun interpret1** —

```
(defun |interpret1| (x rootMode posnForm)
 (let (node modeSet newRootMode argVal val)
 (declare (special |$genValue| |$eval|))
```

```
  (setq node (|mkAtreeWithSrcPos| x posnForm))
  (when rootMode (|putTarget| node rootMode))
  (setq modeSet (|bottomUp| node))
  (if (null |$eval|)
   modeSet
   (progn
    (if (null rootMode)
     (setq newRootMode (car modeSet))
     (setq newRootMode rootMode))
    (setq argVal (|getArgValue| node newRootMode))
    (cond
     ((and argVal (null |$genValue|))
       (|objNew| argVal newRootMode))
     ((and argVal (setq val (|getValue| node)))
       (|interpret2| val newRootMode posnForm))
     (t
       (|keyedSystemError| 'S2IS0053 (list x))))))))
```

———-

5.3.48 defun interpret2

This is the late interpretCoerce. I removed the call to coerceInteractive, so it only does the JENKS cases ALBI [objVal p**??**]
[objMode p**??**]
[member p1042]
[objNew p**??**]
[systemErrorHere p**??**]
[coerceInteractive p**??**]
[throwKeyedMsgCannotCoerceWithValue p**??**]
[$EmptyMode p**??**]
[$ThrowAwayMode p**??**]

— **defun interpret2** —

```
(defun |interpret2| (object m1 posnForm)
 (declare (ignore  posnForm))
 (let (x m op ans)
 (declare (special |$EmptyMode| |$ThrowAwayMode|))
  (cond
   ((equal m1 |$ThrowAwayMode|) object)
   (t
    (setq x (|objVal| object))
    (setq m (|objMode| object))
    (cond
     ((equal m |$EmptyMode|)
      (cond
```

```
    ((and (consp x)
          (progn (setq op (qcar x)) t)
          (|member| op '(map stream)))
      (|objNew| x m1))
    ((equal m1 |$EmptyMode|)
      (|objNew| x m))
    (t
      (|systemErrorHere| "interpret2"))))
(m1
 (if (setq ans (|coerceInteractive| object m1))
  ans
  (|throwKeyedMsgCannotCoerceWithValue| x m m1)))
(t object))))))
```

———-

5.3.49 defun Result Output Printing

Prints out the value x which is of type m, and records the changes in environment `$e` into `$InteractiveFrame` `$printAnyIfTrue` is documented in setvart.boot. It is controlled with the `)se me any` command. [output p**??**]
[putHist p586]
[objNewWrap p**??**]
[printTypeAndTime p58]
[printStorage p58]
[printStatisticsSummary p57]
[mkCompanionPage p**??**]
[recordAndPrintTest p**??**]
[$outputMode p**??**]
[$mkTestOutputType p**??**]
[$runTestFlag p**??**]
[$e p**??**]
[$mkTestFlag p**??**]
[$HTCompanionWindowID p52]
[$QuietCommand p47]
[$printStatisticsSummaryIfTrue p741]
[$printTypeIfTrue p743]
[$printStorageIfTrue p**??**]
[$printTimeIfTrue p742]
[$Void p**??**]
[$algebraOutputStream p754]
[$collectOutput p**??**]
[$EmptyMode p**??**]
[$printVoidIfTrue p744]
[$outputMode p**??**]

[$printAnyIfTrue p726]

— **defun recordAndPrint** —

```
(defun |recordAndPrint| (x md)
 (let (|$outputMode| xp mdp mode)
 (declare (special |$outputMode| |$mkTestOutputType| |$runTestFlag| |$e|
                   |$mkTestFlag| |$HTCompanionWindowID| |$QuietCommand|
                   |$printStatisticsSummaryIfTrue| |$printTypeIfTrue|
                   |$printStorageIfTrue| |$printTimeIfTrue| |$Void|
                   |$algebraOutputStream| |$collectOutput| |$EmptyMode|
                   |$printVoidIfTrue| |$outputMode| |$printAnyIfTrue|))
  (cond
   ((and (equal md '(|Any|)) |$printAnyIfTrue|)
    (setq mdp (car x))
    (setq xp (cdr x)))
   (t
    (setq mdp md)
    (setq xp x)))
  (setq |$outputMode| md)
  (if (equal md |$EmptyMode|)
   (setq mode (|quadSch|))
   (setq mode md))
  (when (or (not (equal md |$Void|)) |$printVoidIfTrue|)
    (unless |$collectOutput| (terpri |$algebraOutputStream|))
    (unless |$QuietCommand| (|output| xp mdp)))
  (|putHist| '% '|value| (|objNewWrap| x md) |$e|)
  (when (or |$printTimeIfTrue| |$printTypeIfTrue|)
    (|printTypeAndTime| xp mdp))
  (when |$printStorageIfTrue| (|printStorage|))
  (when |$printStatisticsSummaryIfTrue| (|printStatisticsSummary|))
  (when (integerp |$HTCompanionWindowID|) (|mkCompanionPage| md))
  (cond
   (|$mkTestFlag| (|recordAndPrintTest| md))
   (|$runTestFlag|
    (setq |$mkTestOutputType| md)
    '|done|)
   (t '|done|))))
```

———-

5.3.50 defun printStatisticsSummary

[sayKeyedMsg p327]
[statisticsSummary p**??**]
[$collectOutput p**??**]

— **defun printStatisticsSummary** —

```
(defun |printStatisticsSummary| ()
 (declare (special |$collectOutput|))
 (unless |$collectOutput|
  (|sayKeyedMsg| 'S2GL0017 (list (|statisticsSummary|)))))
```

———-

5.3.51 defun printStorage

[makeLongSpaceString p**??**]
[$interpreterTimedClasses p**??**]
[$collectOutput p**??**]
[$interpreterTimedNames p**??**]

— **defun printStorage** —

```
(defun |printStorage| ()
 (declare (special |$interpreterTimedClasses| |$collectOutput|
                   |$interpreterTimedNames|))
  (unless |$collectOutput|
   (|sayKeyedMsg| 'S2GL0016
    (list
     (|makeLongSpaceString|
       |$interpreterTimedNames|
       |$interpreterTimedClasses|)))))
```

———-

5.3.52 defun printTypeAndTime

[printTypeAndTimeSaturn p60]
[printTypeAndTimeNormal p59]
[$saturn p**??**]

— **defun printTypeAndTime** —

```
(defun |printTypeAndTime| (x m)
 (declare (special |$saturn|))
 (if |$saturn|
  (|printTypeAndTimeSaturn| x m)
  (|printTypeAndTimeNormal| x m)))
```

———-

5.3.53 defun printTypeAndTimeNormal

[retract p1049]
[qcar p??]
[retract p1049]
[objNewWrap p??]
[objMode p??]
[sameUnionBranch p61]
[makeLongTimeString p??]
[msgText p61]
[sayKeyedMsg p327]
[justifyMyType p62]
[$outputLines p??]
[$collectOutput p??]
[$printTypeIfTrue p743]
[$printTimeIfTrue p742]
[$outputLines p??]
[$interpreterTimedNames p??]
[$interpreterTimedClasses p??]

— **defun printTypeAndTimeNormal** —

```
(defun |printTypeAndTimeNormal| (x m)
 (let (xp mp timeString result)
 (declare (special |$outputLines| |$collectOutput| |$printTypeIfTrue|
                   |$printTimeIfTrue| |$outputLines|
                   |$interpreterTimedNames| |$interpreterTimedClasses|))
  (cond
   ((and (consp m) (eq (qcar m) '|Union|))
    (setq xp (|retract| (|objNewWrap| x m)))
    (setq mp (|objMode| xp))
    (setq m
     (cons '|Union|
      (append
       (dolist (arg (qcdr m) (nreverse result))
         (when (|sameUnionBranch| arg mp) (push arg result)))
       (list "..."))))))
  (when |$printTimeIfTrue|
   (setq timeString
    (|makeLongTimeString|
     |$interpreterTimedNames|
     |$interpreterTimedClasses|)))
  (cond
   ((and |$printTimeIfTrue| |$printTypeIfTrue|)
    (if |$collectOutput|
     (push (|msgText| 'S2GL0012 (list m)) |$outputLines|)
     (|sayKeyedMsg| 'S2GL0014 (list m timeString ))))
   (|$printTimeIfTrue|
```

```
    (unless |$collectOutput| (|sayKeyedMsg| 'S2GL0013 (list timeString))))
   (|$printTypeIfTrue|
    (if |$collectOutput|
     (push (|justifyMyType| (|msgText| 'S2GL0012 (list m))) |$outputLines|)
     (|sayKeyedMsg| 'S2GL0012 (list m)))))))
```

———-

5.3.54 defun printTypeAndTimeSaturn

[makeLongTimeString p??]
[form2StringAsTeX p??]
[devaluate p??]
[printAsTeX p61]
[$printTimeIfTrue p742]
[$printTypeIfTrue p743]
[$interpreterTimedClasses p??]
[$interpreterTimedNames p??]

— defun printTypeAndTimeSaturn —

```
(defun |printTypeAndTimeSaturn| (x m)
 (declare (ignore x))
 (let (timeString typeString)
 (declare (special |$printTimeIfTrue| |$printTypeIfTrue|
                   |$interpreterTimedClasses| |$interpreterTimedNames|))
  (if |$printTimeIfTrue|
   (setq timeString
    (|makeLongTimeString|
     |$interpreterTimedNames|
     |$interpreterTimedClasses|))
   (setq timeString ""))
  (if |$printTypeIfTrue|
   (setq typeString (|form2StringAsTeX| (|devaluate| m)))
   (setq typeString ""))
  (when |$printTypeIfTrue|
   (|printAsTeX| "\\axPrintType{")
   (if (consp typeString)
    (mapc #'|printAsTeX| typeString)
    (|printAsTeX| typeString))
   (|printAsTeX| "}"))
  (when |$printTimeIfTrue|
   (|printAsTeX| "\\axPrintTime{")
   (|printAsTeX| timeString)
   (|printAsTeX| "}"))))
```

———-

5.3.55 defun printAsTeX

[$texOutputStream p??]

— **defun printAsTeX 0** —

```
(defun |printAsTeX| (x)
 (declare (special |$texOutputStream|))
 (princ x |$texOutputStream|))
```

———-

5.3.56 defun sameUnionBranch

```
sameUnionBranch(uArg, m) ==
  uArg is [":", ., t] => t = m
  uArg = m
```

— **defun sameUnionBranch 0** —

```
(defun |sameUnionBranch| (uArg m)
 (let (t1 t2 t3)
 (cond
  ((and (consp uArg)
        (eq (qcar uArg) '|:|)
        (progn
         (setq t1 (qcdr uArg))
         (and (consp t1)
              (progn
               (setq t2 (qcdr t1))
               (and (consp t2)
                    (eq (qcdr t2) nil)
                    (progn (setq t3 (qcar t2)) t))))))
   (equal t3 m))
  (t (equal uArg m)))))
```

———-

5.3.57 defun msgText

[segmentKeyedMsg p328]
[getKeyedMsg p327]

[substituteSegmentedMsg p??]
[flowSegmentedMsg p??]
[$linelength p765]
[$margin p765]

— **defun msgText** —

```
(defun |msgText| (key args)
 (let (msg)
 (declare (special $linelength $margin))
  (setq msg (|segmentKeyedMsg| (|getKeyedMsg| key)))
  (setq msg (|substituteSegmentedMsg| msg args))
  (setq msg (|flowSegmentedMsg| msg $linelength $margin))
  (apply #'concat (mapcar #'princ-to-string (cdar msg)))))
```

———-

5.3.58 defun Right-justify the Type output

[fillerSpaces p20]
[$linelength p765]

— **defun justifyMyType** —

```
(defun |justifyMyType| (arg)
 (let (len)
 (declare (special $linelength))
  (setq len (|#| arg))
  (if (> len $linelength)
   arg
   (concat (|fillerSpaces| (- $linelength len)) arg))))
```

———-

5.3.59 defun Destructively fix quotes in strings

[unescapeStringsInForm p62]
[$funnyBacks p??]
[$funnyQuote p??]

— **defun unescapeStringsInForm** —

```
(defun |unescapeStringsInForm| (form)
 (let (str)
```

```
(declare (special |$funnyBacks| |$funnyQuote|))
 (cond
  ((stringp form)
   (setq str (nsubstitute #\" |$funnyQuote| form))
   (nsubstitute #\\ |$funnyBacks| str))
  ((consp form)
   (|unescapeStringsInForm| (car form))
   (|unescapeStringsInForm| (cdr form))
   form)
  (t form))))
```

———-

5.3.60 Include a file into the stream

[ST p??]
[intloopInclude0 p63]

— **defun intloopInclude** —

```
(defun |intloopInclude| (name n)
  "Include a file into the stream"
  (with-open-file (st name) (|intloopInclude0| st name n)))
```

———-

5.3.61 defun intloopInclude0

[incStream p73]
[intloopProcess p64]
[next p38]
[intloopEchoParse p69]
[insertpile p333]
[lineoftoks p108]
[$lines p??]

— **defun intloopInclude0** —

```
(defun |intloopInclude0| (|st| |name| |n|)
 (let (|$lines|)
 (declare (special |$lines|))
  (setq |$lines| (|incStream| |st| |name|))
  (|intloopProcess| |n| NIL
   (|next| #'|intloopEchoParse|
```

```
    (|next| #'|insertpile|
     (|next| #'|lineoftoks|
      |$lines|)))))))
```

5.3.62 defun intloopProcess

[StreamNull p331]
[pfAbSynOp? p410]
[setCurrentLine p41]
[tokPart p411]
[intloopProcess p64]
[intloopSpadProcess p64]
[$systemCommandFunction p??]
[$systemCommandFunction p??]

— defun intloopProcess —

```
(defun |intloopProcess| (n interactive s)
 (let (ptree lines t1)
 (declare (special |$systemCommandFunction|))
  (cond
   ((|StreamNull| s) n)
   (t
    (setq t1 (car s))
    (setq lines (car t1))
    (setq ptree (cadr t1))
    (cond
     ((|pfAbSynOp?| ptree '|command|)
       (when interactive (|setCurrentLine| (|tokPart| ptree)))
       (funcall |$systemCommandFunction| (|tokPart| ptree))
       (|intloopProcess| n interactive (cdr s)))
     (t
       (|intloopProcess|
         (|intloopSpadProcess| n lines ptree interactive)
         interactive (cdr s))))))))
```

5.3.63 defun intloopSpadProcess

[flung p??]
[SpadCompileItem p??]
[intCoerceFailure p32]

[intSpadReader p32]
[ncPutQ p414]
[CatchAsCan p??]
[Catch p??]
[intloopSpadProcess,interp p65]
[$currentCarrier p??]
[$ncMsgList p27]
[$intCoerceFailure p32]
[$intSpadReader p32]
[$prevCarrier p??]
[$stepNo p??]
[$NeedToSignalSessionManager p44]
[flung p??]

— **defun intloopSpadProcess** —

```
(defun |intloopSpadProcess| (stepNo lines ptree interactive?)
 (let (|$stepNo| result cc)
 (declare (special |$stepNo| |$prevCarrier| |$intSpadReader| |flung|
                   |$intCoerceFailure| |$ncMsgList| |$currentCarrier|
                   |$NeedToSignalSessionManager|))
  (setq |$stepNo| stepNo)
  (setq |$currentCarrier| (setq cc (list '|carrier|)))
  (|ncPutQ| cc '|stepNumber| stepNo)
  (|ncPutQ| cc '|messages| |$ncMsgList|)
  (|ncPutQ| cc '|lines| lines)
  (setq |$ncMsgList| nil)
  (setq result
    (catch '|SpadCompileItem|
     (catch |$intCoerceFailure|
      (catch |$intSpadReader|
       (|intloopSpadProcess,interp| cc ptree interactive?)))))
  (setq |$NeedToSignalSessionManager| t)
  (setq |$prevCarrier| |$currentCarrier|)
  (cond
    ((eq result '|ncEnd|) stepNo)
    ((eq result '|ncError|) stepNo)
    ((eq result '|ncEndItem|) stepNo)
    (t (1+ stepNo)))))
```

———-

5.3.64 defun intloopSpadProcess,interp

[ncConversationPhase p68]
[ncEltQ p414]

[ncError p69]

— **defun intloopSpadProcess,interp** —

```
(defun |intloopSpadProcess,interp| (cc ptree interactive?)
 (|ncConversationPhase| #'|phParse| (list cc ptree))
 (|ncConversationPhase| #'|phMacro| (list cc))
 (|ncConversationPhase| #'|phIntReportMsgs| (list cc interactive?))
 (|ncConversationPhase| #'|phInterpret| (list cc))
 (unless (eql (length (|ncEltQ| cc '|messages|)) 0) (|ncError|)))
```

———-

5.3.65 defun phParse

TPDHERE: The pform function has a leading percent sign. fix this

```
phParse: carrier[tokens,...] -> carrier[ptree, tokens,...]
```

[ncPutQ p414]

— **defun phParse** —

```
(defun |phParse| (carrier ptree)
  (|ncPutQ| carrier '|ptree| ptree)
  'ok)
```

———-

5.3.66 defun phIntReportMsgs

```
carrier[lines,messages,..]-> carrier[lines,messages,..]
```

[ncEltQ p414]
[ncPutQ p414]
[processMsgList p367]
[$erMsgToss p??]

— **defun phIntReportMsgs** —

```
(defun |phIntReportMsgs| (carrier interactive?)
 (declare (ignore interactive?))
 (let (nerr msgs lines)
 (declare (special |$erMsgToss|))
```

```
  (cond
   (|$erMsgToss| 'ok)
   (t
     (setq lines (|ncEltQ| carrier '|lines|))
     (setq msgs (|ncEltQ| carrier '|messages|))
     (setq nerr (length msgs))
     (|ncPutQ| carrier '|ok?| (eql nerr 0))
     (cond
      ((eql nerr 0) 'ok)
      (t
       (|processMsgList| msgs lines)
       (|sayKeyedMsg| 'S2CTP010 (list nerr))
       'ok))))))
```

———-

5.3.67 defun phInterpret

[ncEltQ p414]
[intInterpretPform p67]
[ncPutQ p414]

— **defun phInterpret** —

```
(defun |phInterpret| (carrier)
 (let (val ptree)
  (setq ptree (|ncEltQ| carrier '|ptree|))
  (setq val (|intInterpretPform| ptree))
  (|ncPutQ| carrier '|value| val)))
```

———-

5.3.68 defun intInterpretPform

[processInteractive p49]
[zeroOneTran p68]
[pf2Sex p297]

— **defun intInterpretPform** —

```
(defun |intInterpretPform| (pf)
  (|processInteractive| (|zeroOneTran| (|pf2Sex| pf)) pf))
```

———-

5.3.69 defun zeroOneTran

[nsubst p??]

— **defun zeroOneTran 0** —

```
(defun |zeroOneTran| (sex)
 (nsubst '|$EmptyMode| '? sex))
```

———-

5.3.70 defun ncConversationPhase

[ncConversationPhase,wrapup p68]
[$ncMsgList p27]

— **defun ncConversationPhase** —

```
(defun |ncConversationPhase| (fn args)
 (let (|$ncMsgList| carrier)
 (declare (special |$ncMsgList|))
  (setq carrier (car args))
  (setq |$ncMsgList| nil)
  (unwind-protect
   (apply fn args)
   (|ncConversationPhase,wrapup| carrier))))
```

———-

5.3.71 defun ncConversationPhase,wrapup

[$ncMsgList p27]

— **defun ncConversationPhase,wrapup** —

```
(defun |ncConversationPhase,wrapup| (carrier)
 (declare (special |$ncMsgList|))
  ((lambda (Var5 m)
   (loop
    (cond
     ((or (atom Var5) (progn (setq m (car Var5)) nil))
       (return nil))
     (t
      (|ncPutQ| carrier '|messages| (cons m (|ncEltQ| carrier '|messages|)))))
    (setq Var5 (cdr Var5))))
```

```
   |$ncMsgList| nil))
```

———-

5.3.72 defun ncError

[SpadCompileItem p??]

— **defun ncError 0** —

```
(defun |ncError| ()
 (throw '|SpadCompileItem| '|ncError|))
```

———-

5.3.73 defun intloopEchoParse

[ncloopDQlines p71]
[setCurrentLine p41]
[mkLineList p70]
[ncloopPrintLines p70]
[npParse p139]
[dqToList p342]
[$EchoLines p??]
[$lines p??]

— **defun intloopEchoParse** —

```
(defun |intloopEchoParse| (s)
 (let (cudr lines stream dq t1)
 (declare (special |$EchoLines| |$lines|))
  (setq t1 (car s))
  (setq dq (car t1))
  (setq stream (cadr t1))
  (setq t1 (|ncloopDQlines| dq |$lines|))
  (setq lines (car t1))
  (setq cudr (cadr t1))
  (|setCurrentLine| (|mkLineList| lines))
  (when |$EchoLines| (|ncloopPrintLines| lines))
  (setq |$lines| cudr)
  (cons (list (list lines (|npParse| (|dqToList| dq)))) (cdr s))))
```

———-

5.3.74 defun ncloopPrintLines

```
;ncloopPrintLines lines ==
;        for line in lines repeat WRITE_-LINE CDR line
;        WRITE_-LINE '" "
```

— defun ncloopPrintLines 0 —

```
(defun |ncloopPrintLines| (lines)
 ((lambda (Var4 line)
  (loop
   (cond
    ((or (atom Var4) (progn (setq line (car Var4)) nil))
      (return nil))
    (t (write-line (cdr line))))
   (setq Var4 (cdr Var4))))
  lines nil)
 (write-line " "))
```

———-

5.3.75 defun mkLineList

```
;mkLineList lines ==
;  l := [CDR line for line in lines | nonBlank CDR line]
;  #l = 1 => CAR l
;  l
```

— defun mkLineList —

```
(defun |mkLineList| (lines)
 (let (l)
  (setq l
   ((lambda (Var2 Var1 line)
     (loop
      (cond
       ((or (atom Var1) (progn (setq line (car Var1)) nil))
         (return (nreverse Var2)))
       (t
        (and (|nonBlank| (cdr line))
             (setq Var2 (cons (cdr line) Var2)))))
      (setq Var1 (cdr Var1))))
    nil lines nil))
  (cond
   ((eql (length l) 1) (car l))
```

```
   (t l))))
```

———-

5.3.76 defun nonBlank

```
;nonBlank str ==
;  value := false
;  for i in 0..MAXINDEX str repeat
;    str.i ^= char " " =>
;      value := true
;      return value
;  value
```

— **defun nonBlank 0** —

```
(defun |nonBlank| (str)
 (let (value)
  ((lambda (Var3 i)
   (loop
    (cond
     ((> i Var3) (return nil))
     (t
      (cond
       ((not (equal (elt str i) (|char| '| |)))
         (identity (progn (setq value t) (return value)))))))
    (setq i (+ i 1))))
   (maxindex str) 0)
  value))
```

———-

5.3.77 defun ncloopDQlines

[StreamNull p331]
[poGlobalLinePosn p72]
[tokPosn p411]
[streamChop p72]

— **defun ncloopDQlines** —

```
(defun |ncloopDQlines| (dq stream)
 (let (b a)
  (|StreamNull| stream)
```

```
  (setq a (|poGlobalLinePosn| (|tokPosn| (cadr dq))))
  (setq b (|poGlobalLinePosn| (caar stream)))
  (|streamChop| (+ (- a b) 1) stream)))
```

———-

5.3.78 defun poGlobalLinePosn

[lnGlobalNum p344]
[poGetLineObject p359]
[ncBug p366]

— defun poGlobalLinePosn —

```
(defun |poGlobalLinePosn| (posn)
 (if posn
  (|lnGlobalNum| (|poGetLineObject| posn))
  (|ncBug| "old style pos objects have no global positions" nil)))
```

———-

5.3.79 defun streamChop

Note that changing the name “lyne” to “line” will break the system. I do not know why. The symptom shows up when there is a file with a large contiguous comment spanning enough lines to overflow the stack. [StreamNull p331]
[streamChop p72]
[ncloopPrefix? p475]

— defun streamChop —

```
(defun |streamChop| (n s)
 (let (d c lyne b a tmp1)
  (cond
   ((|StreamNull| s) (list nil nil))
   ((eql n 0) (list nil s))
   (t
     (setq tmp1 (|streamChop| (- n 1) (cdr s)))
     (setq a (car tmp1))
     (setq b (cadr tmp1))
     (setq lyne (car s))
     (setq c (|ncloopPrefix?| ")command" (cdr lyne)))
     (setq d (cons (car lyne) (cond (c c) (t (cdr lyne)))))
     (list (cons d a) b)))))
```

———-

5.3.80 defun ncloopInclude0

[incStream p73]
[ncloopProcess p??]
[next p38]
[ncloopEchoParse p??]
[insertpile p333]
[lineoftoks p108]
[$lines p??]

— **defun ncloopInclude0** —

```
(defun |ncloopInclude0| (st name n)
 (let (|$lines|)
 (declare (special |$lines|))
   (setq |$lines| (|incStream| st name))
   (|ncloopProcess| n nil
    (|next| #'|ncloopEchoParse|
     (|next| #'|insertpile|
      (|next| #'|lineoftoks|
       |$lines|))))))
```

———-

5.3.81 defun incStream

[incRenumber p74]
[incLude p76]
[incRgen p103]
[Top p77]

— **defun incStream** —

```
(defun |incStream| (st fn)
 (declare (special |Top|))
 (|incRenumber| (|incLude| 0 (|incRgen| st) 0 (list fn) (list |Top|))))
```

———-

5.3.82 defun incRenumber

[incZip p74]
[incIgen p75]

— **defun incRenumber** —

```
(defun |incRenumber| (ssx)
 (|incZip| #'|incRenumberLine| ssx (|incIgen| 0)))
```

————-

5.3.83 defun incZip

[Delay p103]
[incZip1 p74]

— **defun incZip** —

```
(defun |incZip| (g f1 f2)
  (|Delay| #'|incZip1| (list g f1 f2)))
```

————-

5.3.84 defun incZip1

[StreamNull p331]
[incZip p74]

— **defun incZip1** —

```
(defun |incZip1| (&rest z)
 (let (f2 f1 g)
  (setq g (car z))
  (setq f1 (cadr z))
  (setq f2 (caddr z))
  (cond
   ((|StreamNull| f1) |StreamNil|)
   ((|StreamNull| f2) |StreamNil|)
   (t
     (cons
      (funcall g (car f1) (car f2))
      (|incZip| g (cdr f1) (cdr f2)))))))
```

————-

5.3.85 defun incIgen

[Delay p103]
[incIgen1 p75]

— **defun incIgen** —

```
(defun |incIgen| (n)
 (|Delay| #'|incIgen1| (list n)))
```

————-

5.3.86 defun incIgen1

[incIgen p75]

— **defun incIgen1** —

```
(defun |incIgen1| (&rest z)
 (let (n)
  (setq n (car z))
  (setq n (+ n 1))
  (cons n (|incIgen| n))))
```

————-

5.3.87 defun incRenumberLine

[incRenumberItem p76]
[incHandleMessage p76]

— **defun incRenumberLine** —

```
(defun |incRenumberLine| (xl gno)
 (let (l)
  (setq l (|incRenumberItem| (elt xl 0) gno))
  (|incHandleMessage| xl)
  l))
```

————-

5.3.88 defun incRenumberItem

[lnSetGlobalNum p344]

— **defun incRenumberItem** —

```
(defun |incRenumberItem| (f i)
 (let (l)
  (setq l (caar f))
  (|lnSetGlobalNum| l i) f))
```

———-

5.3.89 defun incHandleMessage

[ncSoftError p349]
[ncBug p366]

— **defun incHandleMessage 0** —

```
(defun |incHandleMessage| (x)
 "Message handling for the source includer"
 (let ((msgtype (elt (elt x 1) 1))
       (pos (car (elt x 0)))
       (key (car (elt (elt x 1) 0)))
       (args (cadr (elt (elt x 1) 0))))

 (cond
  ((eq msgtype '|none|)    0)
  ((eq msgtype '|error|)   (|ncSoftError| pos key args))
  ((eq msgtype '|warning|) (|ncSoftError| pos key args))
  ((eq msgtype '|say|)     (|ncSoftError| pos key args))
  (t                       (|ncBug| key args)))))
```

———-

5.3.90 defun incLude

[Delay p103]
[incLude1 p81]

— **defun incLude** —

```
(defun |incLude| (eb ss ln ufos states)
 (|Delay| #'|incLude1| (list eb ss ln ufos states)))
```

———-

5.3.91 defmacro Rest

— **defmacro Rest** —

```
(defmacro |Rest| ()
 "used in incLude1 for parsing; s is not used."
 '(|incLude| eb (cdr ss) lno ufos states))
```

———-

5.3.92 defvar $Top

— **initvars** —

```
(defvar |Top| 1 "used in incLude1 for parsing")
```

———-

5.3.93 defvar $IfSkipToEnd

— **initvars** —

```
(defvar |IfSkipToEnd| 10 "used in incLude1 for parsing")
```

———-

5.3.94 defvar $IfKeepPart

— **initvars** —

```
(defvar |IfKeepPart| 11 "used in incLude1 for parsing")
```

———-

5.3.95 defvar $IfSkipPart

— **initvars** —

```
(defvar |IfSkipPart| 12 "used in incLude1 for parsing")
```

———-

5.3.96 defvar $ElseifSkipToEnd

— **initvars** —

```
(defvar |ElseifSkipToEnd| 20 "used in incLude1 for parsing")
```

———-

5.3.97 defvar $ElseifKeepPart

— **initvars** —

```
(defvar |ElseifKeepPart| 21 "used in incLude1 for parsing")
```

———-

5.3.98 defvar $ElseifSkipPart

— **initvars** —

```
(defvar |ElseifSkipPart| 22 "used in incLude1 for parsing")
```

———-

5.3.99 defvar $ElseSkipToEnd

— **initvars** —

```
(defvar |ElseSkipToEnd| 30 "used in incLude1 for parsing")
```

———-

5.3.100 defvar $ElseKeepPart

— **initvars** —

```
(defvar |ElseKeepPart| 31 "used in incLude1 for parsing")
```

———-

5.3.101 defvar $Top?

[quotient p??]

— **defun Top? 0** —

```
(defun |Top?| (|st|)
 "used in incLude1 for parsing"
 (eql (quotient |st| 10) 0))
```

———-

5.3.102 defvar $If?

[quotient p??]

— **defun If?** —

```
(defun |If?| (|st|)
 "used in incLude1 for parsing"
 (eql (quotient |st| 10) 1))
```

———-

5.3.103 defvar $Elseif?

[QUOTIENT p??]

— **defun Elseif?** —

```
(defun |Elseif?| (|st|)
 "used in incLude1 for parsing"
 (eql (quotient |st| 10) 2))
```

———-

5.3.104 defvar $Else?

[QUOTIENT p??]

— **defun Else?** —

```
(defun |Else?| (|st|)
 "used in incLude1 for parsing"
 (eql (quotient |st| 10) 3))
```

———-

5.3.105 defvar $SkipEnd?

[remainder p??]

— **defun SkipEnd?** —

```
(defun |SkipEnd?| (|st|)
 "used in incLude1 for parsing"
 (eql (remainder |st| 10) 0))
```

———-

5.3.106 defvar $KeepPart?

[remainder p??]

— **defun KeepPart?** —

```
(defun |KeepPart?| (|st|)
 "used in incLude1 for parsing"
 (eql (remainder |st| 10) 1))
```

———-

5.3.107 defvar $SkipPart?

[remainder p??]

— **defun SkipPart?** —

```
(defun |SkipPart?| (|st|)
 "used in incLude1 for parsing"
 (eql (remainder |st| 10) 2))
```

———-

5.3.108 defvar $Skipping?

[KeepPart? p80]

— **defun Skipping?** —

```
(defun |Skipping?| (|st|)
 "used in incLude1 for parsing"
 (null (|KeepPart?| |st|)))
```

———-

5.3.109 defun incLude1

[StreamNull p331]
[Top? p79]
[xlPrematureEOF p86]
[Skipping? p81]
[xlSkip p89]
[Rest p77]
[xlOK p86]
[xlOK1 p86]
[concat p1041]
[incCommandTail p101]
[xlSay p89]
[xlNoSuchFile p90]
[xlCannotRead p91]
[incActive? p103]
[xlFileCycle p92]
[incLude p76]
[incFileInput p102]

[incAppend p87]
[inclFname p102]
[xlConActive p93]
[xlConStill p94]
[incConsoleInput p102]
[incNConsoles p103]
[xlConsole p94]
[xlSkippingFin p95]
[xlPrematureFin p95]
[assertCond p96]
[ifCond p89]
[If? p79]
[Elseif? p79]
[xlIfSyntax p96]
[SkipEnd? p80]
[KeepPart? p80]
[SkipPart? p81]
[xlIfBug p97]
[xlCmdBug p98]
[expand-tabs p**??**]
[incClassify p99]

— defun incLude1 —

```
(defun |incLude1| (&rest z)
 (let (pred s1 n tail head includee fn1 info str state lno states
       ufos ln ss eb)
  (setq eb (car z))
  (setq ss (cadr . (z)))
  (setq ln (caddr . (z)))
  (setq ufos (cadddr . (z)))
  (setq states (car (cddddr . (z))))
  (setq lno (+ ln 1))
  (setq state (elt states 0))
  (cond
   ((|StreamNull| ss)
     (cond
      ((null (|Top?| state))
       (cons (|xlPrematureEOF| eb ")--premature end" lno ufos)
             |StreamNil|))
      (t |StreamNil|)))
   (t
    (progn
     (setq str (expand-tabs (car ss)))
     (setq info (|incClassify| str))
     (cond
      ((null (elt info 0))
       (cond
```

```
  ((|Skipping?| state)
   (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
  (t
   (cons (|xlOK| eb str lno (elt ufos 0)) (|Rest|)))))
((equal (elt info 2) "other")
 (cond
  ((|Skipping?| state)
   (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
  (t
   (cons
    (|xlOK1| eb str (concat ")command" str) lno (elt ufos 0))
    (|Rest|)))))
((equal (elt info 2) "say")
 (cond
  ((|Skipping?| state)
   (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
  (t
   (progn
    (setq str (|incCommandTail| str info))
    (cons (|xlSay| eb str lno ufos str)
     (cons (|xlOK| eb str lno (ELT ufos 0)) (|Rest|)))))))
((equal (elt info 2) "include")
 (cond
  ((|Skipping?| state)
   (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
  (t
   (progn
    (setq fn1 (|inclFname| str info))
    (cond
     ((null fn1)
      (cons (|xlNoSuchFile| eb str lno ufos fn1) (|Rest|)))
     ((null (probe-file fn1))
      (cons (|xlCannotRead| eb str lno ufos fn1) (|Rest|)))
     ((|incActive?| fn1 ufos)
      (cons (|xlFileCycle| eb str lno ufos fn1) (|Rest|)))
     (t
      (progn
       (setq includee
        (|incLude| (+ eb (elt info 1))
                   (|incFileInput| fn1)
                   0
                   (cons fn1 ufos)
                   (cons |Top| states)))
       (cons (|xlOK| eb str lno (elt ufos 0))
             (|incAppend| includee (|Rest|)))))))))))
((equal (elt info 2) "console")
 (cond
  ((|Skipping?| state)
   (cons (|xlSkip| eb str lno (elt ufos 0)) (|Rest|)))
  (t
```

```
   (progn
    (setq head
     (|incLude| (+ eb (elt info 1))
                (|incConsoleInput|)
                0
                (cons "console" ufos)
                (cons |Top| states)))
    (setq tail (|Rest|))
    (setq n (|incNConsoles| ufos))
    (cond
     ((< 0 n)
      (setq head
       (cons (|xlConActive| eb str lno ufos n) head))
      (setq tail
       (cons (|xlConStill| eb str lno ufos n) tail))))
    (setq head (cons (|xlConsole| eb str lno ufos) head))
    (cons (|xlOK| eb str lno (elt ufos 0))
          (|incAppend| head tail))))))
((equal (elt info 2) "fin")
 (cond
  ((|Skipping?| state)
   (cons (|xlSkippingFin| eb str lno ufos) (|Rest|)))
  ((null (|Top?| state))
   (cons (|xlPrematureFin| eb str lno ufos) |StreamNil|))
  (t
   (cons (|xlOK| eb str lno (elt ufos 0)) |StreamNil|))))
((equal (elt info 2) "assert")
 (cond
  ((|Skipping?| state)
   (cons (|xlSkippingFin| eb str lno ufos) (|Rest|)))
  (t
   (progn
    (|assertCond| str info)
    (cons (|xlOK| eb str lno (elt ufos 0))
          (|incAppend| includee (|Rest|)))))))
((equal (elt info 2) "if")
 (progn
  (setq s1
   (cond
    ((|Skipping?| state) |IfSkipToEnd|)
    (t
     (cond
      ((|ifCond| str info) |IfKeepPart|)
      (t |IfSkipPart|)))))
  (cons (|xlOK| eb str lno (elt ufos 0))
        (|incLude| eb (cdr ss) lno ufos (cons s1 states)))))
((equal (elt info 2) "elseif")
 (cond
  ((and (null (|If?| state)) (null (|Elseif?| state)))
   (cons (|xlIfSyntax| eb str lno ufos info states)
```

```
          |StreamNil|))
  (t
   (cond
    ((or (|SkipEnd?| state)
         (|KeepPart?| state)
         (|SkipPart?| state))
     (setq s1
      (cond
       ((|SkipPart?| state)
        (setq pred (|ifCond| str info))
        (cond
         (pred |ElseifKeepPart|)
         (t |ElseifSkipPart|)))
       (t |ElseifSkipToEnd|)))
     (cons (|xlOK| eb str lno (elt ufos 0))
           (|incLude| eb (cdr ss) lno ufos (cons s1 (cdr states)))))
    (t
     (cons (|xlIfBug| eb str lno ufos) |StreamNil|))))))
((equal (elt info 2) "else")
 (cond
  ((and (null (|If?| state)) (null (|Elseif?| state)))
   (cons (|xlIfSyntax| eb str lno ufos info states)
         |StreamNil|))
  (t
   (cond
    ((or (|SkipEnd?| state)
         (|KeepPart?| state)
         (|SkipPart?| state))
      (setq s1
       (cond ((|SkipPart?| state) |ElseKeepPart|) (t |ElseSkipToEnd|)))
      (cons (|xlOK| eb str lno (elt ufos 0))
            (|incLude| eb (cdr ss) lno ufos (cons s1 (cdr states)))))
    (t
     (cons (|xlIfBug| eb str lno ufos) |StreamNil|))))))
((equal (elt info 2) "endif")
 (cond
  ((|Top?| state)
   (cons (|xlIfSyntax| eb str lno ufos info states)
         |StreamNil|))
  (t
   (cons (|xlOK| eb str lno (elt ufos 0))
         (|incLude| eb (cdr ss) lno ufos (cdr states))))))
(t (cons (|xlCmdBug| eb str lno ufos) |StreamNil|))))))))
```

———-

5.3.110 defun xlPrematureEOF

[xlMsg p86]
[inclmsgPrematureEOF p88]

— **defun xlPrematureEOF** —

```
(defun |xlPrematureEOF| (eb str lno ufos)
 (|xlMsg| eb str lno (elt ufos 0)
          (list (|inclmsgPrematureEOF| (elt ufos 0)) '|error|)))
```

———-

5.3.111 defun xlMsg

[incLine p87]

— **defun xlMsg** —

```
(defun |xlMsg| (extrablanks string localnum fileobj mess)
 (let ((globalnum -1))
  (list (incLine extrablanks string globalnum localnum fileobj) mess)))
```

———-

5.3.112 defun xlOK

[lxOK1 p**??**]

— **defun xlOK** —

```
(defun |xlOK| (extrablanks string localnum fileobj)
 (|xlOK1| extrablanks string string localnum fileobj))
```

———-

5.3.113 defun xlOK1

[incLine1 p88]

— **defun xlOK1** —

```
(defun |xlOK1| (extrablanks string string1 localnum fileobj)
 (let ((globalnum -1))
  (list (incLine1 extrablanks string string1 globalnum localnum fileobj)
    (list nil '|none|))))
```

———-

5.3.114 defun incAppend

[Delay p103]
[incAppend1 p87]

— **defun incAppend** —

```
(defun |incAppend| (x y)
 (|Delay| #'|incAppend1| (list x y)))
```

———-

5.3.115 defun incAppend1

[StreamNull p331]
[incAppend p87]

— **defun incAppend1** —

```
(defun |incAppend1| (&rest z)
 (let (y x)
  (setq x (car z))
  (setq y (cadr z))
  (cond
   ((|StreamNull| x)
    (cond ((|StreamNull| y) |StreamNil|) (t y)))
   (t
    (cons (car x) (|incAppend| (cdr x) y))))))
```

———-

5.3.116 defun incLine

[incLine1 p88]

— **defun incLine** —

```
(defun incLine (extrablanks string globalnum localnum fileobj)
 (incLine1 extrablanks string string globalnum localnum fileobj))
```

———-

5.3.117 defun incLine1

[lnCreate p343]

— **defun incLine1** —

```
(defun incLine1 (extrablanks string string1 globalnum localnum fileobj)
 (cons
  (cons (|lnCreate| extrablanks string globalnum localnum fileobj) 1) string1))
```

———-

5.3.118 defun inclmsgPrematureEOF

[origin p**??**]

— **defun inclmsgPrematureEOF 0** —

```
(defun |inclmsgPrematureEOF| (ufo)
 (list 'S2CI0002 (list (|theorigin| ufo))))
```

———-

5.3.119 defun theorigin

— **defun theorigin 0** —

```
(defun |theorigin| (x) (list #'|porigin| x))
```

———-

5.3.120 defun porigin

[stringp p**??**]

— **defun porigin** —

```
(defun |porigin| (x)
 (if (stringp x)
  x
  (|pfname| x)))
```

———-

5.3.121 defun ifCond

[MakeSymbol p**??**]
[incCommandTail p101]
[$inclAssertions p**??**]

— **defun ifCond** —

```
(defun |ifCond| (s info)
 (let (word)
 (declare (special |$inclAssertions|))
  (setq word
   (|MakeSymbol| (string-trim *whitespace* (|incCommandTail| s info))))
  (member word |$inclAssertions|)))
```

———-

5.3.122 defun xlSkip

[incLine p87]
[CONCAT p**??**]

— **defun xlSkip** —

```
(defun |xlSkip| (extrablanks str localnum fileobj)
 (let ((string (concat "-- Omitting:" str)) (globalnum -1))
 (list
  (incLine extrablanks string globalnum localnum fileobj)
  (list nil '|none|))))
```

———-

5.3.123 defun xlSay

[xlMsg p86]
[inclmsgSay p90]

— **defun xlSay** —

```
(defun |xlSay| (eb str lno ufos x)
 (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgSay| x) '|say|)))
```

————-

5.3.124 defun inclmsgSay

[id p??]

— **defun inclmsgSay** —

```
(defun |inclmsgSay| (str)
 (list 'S2CI0001 (list (|theid| str))))
```

————-

5.3.125 defun theid

— **defun theid 0** —

```
(defun |theid| (a) (list identity a))
```

————-

5.3.126 defun xlNoSuchFile

[xlMsg p86]
[inclmsgNoSuchFile p91]

— **defun xlNoSuchFile** —

```
(defun |xlNoSuchFile| (eb str lno ufos fn)
 (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgNoSuchFile| fn) '|error|)))
```

————-

5.3.127 defun inclmsgNoSuchFile

[thefname p91]

— **defun inclmsgNoSuchFile** —

```
(defun |inclmsgNoSuchFile| (fn)
 (list 'S2CI0010 (list (|thefname| fn))))
```

————-

5.3.128 defun thefname

[pfname p91]

— **defun thefname 0** —

```
(defun |thefname| (x) (list #'|pfname| x))
```

————-

5.3.129 defun pfname

[PathnameString p??]

— **defun pfname** —

```
(defun |pfname| (x) (|PathnameString| x))
```

————-

5.3.130 defun xlCannotRead

[xlMsg p86]
[inclmsgCannotRead p92]

— **defun xlCannotRead** —

```
(defun |xlCannotRead| (eb str lno ufos fn)
 (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgCannotRead| fn) '|error|)))
```

————-

5.3.131 defun inclmsgCannotRead

[thefname p91]

— **defun inclmsgCannotRead** —

```
(defun |inclmsgCannotRead| (fn)
 (list 'S2CI0011 (list (|thefname| fn))))
```

————-

5.3.132 defun xlFileCycle

[xlMsg p86]
[inclmsgFileCycle p92]

— **defun xlFileCycle** —

```
(defun |xlFileCycle| (eb str lno ufos fn)
 (|xlMsg| eb str lno (elt ufos 0)
  (list (|inclmsgFileCycle| ufos fn) '|error|)))
```

————-

5.3.133 defun inclmsgFileCycle

```
;inclmsgFileCycle(ufos,fn) ==
;    flist := [porigin n for n in reverse ufos]
;    f1    := porigin fn
;    cycle := [:[:[n,'"==>"] for n in flist], f1]
;    ['S2CI0004, [%id cycle, %id f1] ]
```

[porigin p88]
[id p??]

— **defun inclmsgFileCycle** —

```
(defun |inclmsgFileCycle| (ufos fn)
 (let (cycle f1 flist)
  (setq flist
   ((lambda (Var8 Var7 n)
     (loop
      (cond
```

```
      ((or (atom Var7) (progn (setq n (car Var7)) nil))
       (return (nreverse Var8)))
      (t
       (setq Var8 (cons (|porigin| n) Var8))))
     (setq Var7 (cdr Var7))))
   nil (reverse ufos) nil))
 (setq f1 (|porigin| fn))
 (setq cycle
  (append
   ((lambda (Var10 Var9 n)
     (loop
      (cond
       ((or (atom Var9) (progn (setq n (car Var9)) nil))
        (return (nreverse Var10)))
       (t
        (setq Var10 (append (reverse (list n "==>")) Var10))))
      (setq Var9 (cdr Var9))))
    nil flist nil)
   (cons f1 nil)))
 (list 'S2CI0004 (list (|theid| cycle) (|theid| f1)))))
```

———-

5.3.134 defun xlConActive

[xlMsg p86]
[inclmsgConActive p93]

— **defun xlConActive** —

```
(defun |xlConActive| (eb str lno ufos n)
 (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgConActive| n) '|warning|)))
```

———-

5.3.135 defun inclmsgConActive

[id p??]

— **defun inclmsgConActive** —

```
(defun |inclmsgConActive| (n)
 (list 'S2CI0006 (list (|theid| n))))
```

———-

5.3.136 defun xlConStill

[xlMsg p86]
[inclmsgConStill p94]

— **defun xlConStill** —

```
(defun |xlConStill| (eb str lno ufos n)
 (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgConStill| n) '|say|)))
```

————-

5.3.137 defun inclmsgConStill

[id p??]

— **defun inclmsgConStill** —

```
(defun |inclmsgConStill| (n)
 (list 'S2CI0007 (list (|theid| n))))
```

————-

5.3.138 defun xlConsole

[xlMsg p86]
[inclmsgConsole p94]

— **defun xlConsole** —

```
(defun |xlConsole| (eb str lno ufos)
 (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgConsole|) '|say|)))
```

————-

5.3.139 defun inclmsgConsole

— **defun inclmsgConsole 0** —

```
(defun |inclmsgConsole| ()
 (list 'S2CI0005 nil))
```

————-

5.3.140 defun xlSkippingFin

[xlMsg p86]
[inclmsgFinSkipped p95]

— **defun xlSkippingFin** —

```
(defun |xlSkippingFin| (eb str lno ufos)
 (|xlMsg| eb str lno (elt ufos 0)
   (list (|inclmsgFinSkipped|) '|warning|)))
```

————-

5.3.141 defun inclmsgFinSkipped

— **defun inclmsgFinSkipped 0** —

```
(defun |inclmsgFinSkipped| ()
 (list 'S2CI0008 nil))
```

————-

5.3.142 defun xlPrematureFin

[xlMsg p86]
[inclmsgPrematureFin p95]

— **defun xlPrematureFin** —

```
(defun |xlPrematureFin| (eb str lno ufos)
 (|xlMsg| eb str lno (elt ufos 0)
   (list (|inclmsgPrematureFin| (elt ufos 0)) '|error|)))
```

————-

5.3.143 defun inclmsgPrematureFin

[origin p??]

— **defun inclmsgPrematureFin** —

```
(defun |inclmsgPrematureFin| (ufo)
 (list 'S2CI0003 (list (|theorigin| ufo))))
```

———-

5.3.144 defun assertCond

[MakeSymbol p**??**]
[incCommandTail p101]
[$inclAssertions p**??**]
[*whitespace* p24]

— **defun assertCond** —

```
(defun |assertCond| (s info)
 (let (word)
 (declare (special |$inclAssertions| *whitespace*))
  (setq word
   (|MakeSymbol| (string-trim *whitespace* (|incCommandTail| s info))))
  (unless (member word |$inclAssertions|)
    (setq |$inclAssertions| (cons word |$inclAssertions|)))))
```

———-

5.3.145 defun xlIfSyntax

[Top? p79]
[Else? p80]
[xlMsg p86]
[inclmsgIfSyntax p97]

— **defun xlIfSyntax** —

```
(defun |xlIfSyntax| (eb str lno ufos info sts)
 (let (context found st)
  (setq st (elt sts 0))
  (setq found (elt info 2))
  (setq context
   (cond
    ((|Top?| st) '|not in an )if...)endif|)
    ((|Else?| st) '|after an )else|)
    (t '|but can't figure out where|)))
  (|xlMsg| eb str lno (elt ufos 0)
    (list (|inclmsgIfSyntax| (elt ufos 0) found context) '|error|))))
```

———-

5.3.146 defun inclmsgIfSyntax

[concat p1041]
[id p??]
[origin p??]

— **defun inclmsgIfSyntax** —

```
(defun |inclmsgIfSyntax| (ufo found context)
 (setq found (concat ")" found))
 (list 'S2CI0009 (list (|theid| found)
                       (|theid| context)
                       (|theorigin| ufo))))
```

———-

5.3.147 defun xlIfBug

[xlMsg p86]
[inclmsgIfBug p97]

— **defun xlIfBug** —

```
(defun |xlIfBug| (eb str lno ufos)
 (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgIfBug|) '|bug|)))
```

———-

5.3.148 defun inclmsgIfBug

— **defun inclmsgIfBug 0** —

```
(defun |inclmsgIfBug| ()
  (list 'S2CB0002 nil))
```

———-

5.3.149 defun xlCmdBug

[xlMsg p86]
[inclmsgCmdBug p98]

— **defun xlCmdBug** —

```
(defun |xlCmdBug| (eb str lno ufos)
 (|xlMsg| eb str lno (elt ufos 0) (list (|inclmsgCmdBug|) '|bug|)))
```

————-

5.3.150 defun inclmsgCmdBug

— **defun inclmsgCmdBug 0** —

```
(defun |inclmsgCmdBug| ()
 (list 'S2CB0003 nil))
```

————-

5.3.151 defvar $incCommands

This is a list of commands that can be in an include file

— **postvars** —

```
(eval-when (eval load)
(setq |incCommands|
 (list "say" "include" "console" "fin" "assert" "if" "elseif" "else" "endif")))
```

————-

5.3.152 defvar $pfMacros

The $pfMacros variable is an alist [[id, state, body-pform], ...] where state is one of: mbody, mparam, mlambda

User-defined macros are maintained in a stack of definitions. This is the stack sequence resulting from the command lines:

```
a ==> 3
a ==> 4
```

```
b ==> 7
(
 (|b| |mbody| ((|integer| (|posn| (0 "b ==> 7" 1 1 "strings") . 6)) . "7"))
 (|a| |mbody| ((|integer| (|posn| (0 "a ==> 4" 1 1 "strings") . 6)) . "4"))
 (|a| |mbody| ((|integer| (|posn| (0 "a ==> 3" 1 1 "strings") . 6)) . "3"))
)
```

— **initvars** —

```
(defvar |$pfMacros| nil)
```

————-

5.3.153 defun incClassify

```
;incClassify(s) ==
;            not incCommand? s => [false,0, '""]
;            i := 1; n := #s
;            while i < n and s.i = char " " repeat i := i + 1
;            i >= n => [true,0,'"other"]
;            eb := (i = 1 => 0; i)
;            bad:=true
;            for p in incCommands while bad repeat
;                incPrefix?(p, i, s) =>
;                    bad:=false
;                    p1 :=p
;            if bad then [true,0,'"other"] else [true,eb,p1]
```

[incCommand? p100]
[incCommands p98]

— **defun incClassify** —

```
(defun |incClassify| (s)
 (let (p1 bad eb n i)
  (declare (special |incCommands|))
  (if (null (|incCommand?| s))
   (list nil 0 "")
   (progn
    (setq i 1)
    (setq n (length s))
    ((lambda ()
      (loop
       (cond
        ((not (and (< i n) (char= (elt s i) #\space)))
         (return nil))
```

```
       (t (setq i (1+ i)))))))
   (cond
    ((not (< i n)) (list t 0 "other"))
    (t
     (if (= i 1)
      (setq eb 0)
      (setq eb i))
     (setq bad t)
     ((lambda (tmp1 p)
       (loop
        (cond
         ((or (atom tmp1)
              (progn (setq p (car tmp1)) nil)
              (not bad))
          (return nil))
         (t
          (cond
           ((|incPrefix?| p i s)
            (identity
             (progn
              (setq bad nil)
              (setq p1 p)))))
          (setq tmp1 (cdr tmp1))))))
      |incCommands| nil)
     (if bad
      (list t 0 "other")
      (list t eb p1))))))
```

———-

5.3.154 defun incCommand?

[char p??]

— **defun incCommand? 0** —

```
(defun |incCommand?| (s)
  "does this start with a close paren?"
  (and (< 0 (length s)) (equal (elt s 0) (|char| '|)|))))
```

———-

5.3.155 defun incPrefix?

```
;incPrefix?(prefix, start, whole) ==
```

```
;            #prefix > #whole-start => false
;            good:=true
;            for i in 0..#prefix-1 for j in start.. while good repeat
;                good:= prefix.i = whole.j
;            good
```

— **defun incPrefix? 0** —

```
(defun |incPrefix?| (prefix start whole)
 (let (good)
  (cond
   ((< (- (length whole) start) (length prefix)) nil)
   (t
    (setq good t)
    ((lambda (Var i j)
      (loop
       (cond
        ((or (> i Var) (not good)) (return nil))
        (t (setq good (equal (elt prefix i) (elt whole j)))))
       (setq i (+ i 1))
       (setq j (+ j 1))))
      (- (length prefix) 1) 0 start)
    good))))
```

———-

5.3.156 defun incCommandTail

[incDrop p101]

— **defun incCommandTail** —

```
(defun |incCommandTail| (s info)
 (let ((start (elt info 1)))
  (when (= start 0) (setq start 1))
  (|incDrop| (+ start (length (elt info 2)) 1) s)))
```

———-

5.3.157 defun incDrop

[substring p**??**]

— **defun incDrop 0** —

```
(defun |incDrop| (n b)
 (if (>= n (length b))
  '||
  (substring b n nil)))
```

———-

5.3.158 defun inclFname

[incFileName p618]
[incCommandTail p101]

— **defun inclFname** —

```
(defun |inclFname| (s info)
 (|incFileName| (|incCommandTail| s info)))
```

———-

5.3.159 defun incFileInput

[incRgen p103]
[make-instream p971]

— **defun incFileInput** —

```
(defun |incFileInput| (fn)
  (|incRgen| (make-instream fn)))
```

———-

5.3.160 defun incConsoleInput

[incRgen p103]
[make-instream p971]

— **defun incConsoleInput** —

```
(defun |incConsoleInput| ()
  (|incRgen| (make-instream 0)))
```

———-

5.3.161 defun incNConsoles

[incNConsoles p103]

— **defun incNConsoles** —

```
(defun |incNConsoles| (ufos)
 (let ((a (member "console" ufos)))
  (if a
   (+ 1 (|incNConsoles| (cdr a)))
   0)))
```

———-

5.3.162 defun incActive?

— **defun incActive? 0** —

```
(defun |incActive?| (fn ufos)
 (member fn ufos))
```

———-

5.3.163 defun incRgen

Note that incRgen1 recursively calls this function. [Delay p103]
[incRgen1 p104]

— **defun incRgen** —

```
(defun |incRgen| (s)
 (|Delay| #'|incRgen1| (list s)))
```

———-

5.3.164 defun Delay

— **defun Delay 0** —

```
(defun |Delay| (f x)
 (cons '|nonnullstream| (cons f x)))
```

———-

5.3.165 defvar $StreamNil

— **initvars** —

```
(defvar |StreamNil| (list '|nullstream|))
```

———-

5.3.166 defvar $StreamNil

— **postvars** —

```
(eval-when (eval load)
  (setq |StreamNil| (list '|nullstream|)))
```

———-

5.3.167 defun incRgen1

This function reads a line from the stream and then conses it up with a recursive call to incRgen. Note that incRgen recursively wraps this function in a delay list. [incRgen p103] [StreamNil p104]

— **defun incRgen1** —

```
(defun |incRgen1| (&rest z)
 (let (a s)
 (declare (special |StreamNil|))
  (setq s (car z))
  (setq a (read-line s nil nil))
  (if (null a)
   (progn
    (close s)
    |StreamNil|)
   (cons a (|incRgen| s)))))
```

———-

Chapter 6

The Token Scanner

6.0.168 defvar $scanKeyWords

— **postvars** —

```
(eval-when (eval load)
(defvar |scanKeyWords|
 (list
  (list "add" 'add)
  (list "and" 'and)
  (list "break" 'break)
  (list "by" 'by)
  (list "case" 'case)
  (list "default" 'default)
  (list "define" 'defn)
  (list "do" 'do)
  (list "else" 'else)
  (list "exit" 'exit)
  (list "export" 'export)
  (list "for" 'for)
  (list "free" 'free)
  (list "from" 'from)
  (list "has" 'has)
  (list "if" 'if)
  (list "import" 'import)
  (list "in" 'in)
  (list "inline" 'inline)
  (list "is" 'is)
  (list "isnt" 'isnt)
  (list "iterate" 'iterate)
  (list "local" '|local|)
  (list "macro" 'macro)
```

```
(list "mod" 'mod)
(list "or" 'or)
(list "pretend" 'pretend)
(list "quo" 'quo)
(list "rem" 'rem)
(list "repeat" 'repeat)
(list "return" 'return)
(list "rule" 'rule)
(list "then" 'then)
(list "where" 'where)
(list "while" 'while)
(list "with" 'with)
(list "|" 'bar)
(list "." 'dot)
(list "::" 'coerce)
(list ":" 'colon)
(list ":-" 'colondash)
(list "@" 'at)
(list "@@" 'atat)
(list "," 'comma)
(list ";" 'semicolon)
(list "**" 'power)
(list "*" 'times)
(list "+" 'plus)
(list "-" 'minus)
(list "<" 'lt)
(list ">" 'gt)
(list "<=" 'le)
(list ">=" 'ge)
(list "=" 'equal)
(list "~=" 'notequal)
(list "~" '~)
(list "^" 'carat)
(list ".." 'seg)
(list "#" '|#|)
(list "&" 'ampersand)
(list "$" '$)
(list "/" 'slash)
(list "\\" 'backslash)
(list "//" 'slashslash)
(list "\\\\" 'backslashbackslash)
(list "/\\" 'slashbackslash)
(list "\\/" 'backslashslash)
(list "=>" 'exit)
(list ":=" 'becomes)
(list "==" 'def)
(list "==>" 'mdef)
(list "->" 'arrow)
(list "<-" 'larrow)
(list "+->" 'gives)
```

```
  (list "(" '|(|)
  (list ")" '|)|)
  (list "(|" '|(\||)
  (list "|)" '|\|)|)
  (list "[" '[)
  (list "]" '])
  (list "[_]" '[])
  (list "{" '{)
  (list "}" '})
  (list "{_}" '{})
  (list "[|" '|[\||)
  (list "|]" '|\|]|)
  (list "[|_|]" '|[\|\|]|)
  (list "{|" '|{\||)
  (list "|}" '|\|}|)
  (list "{|_|}" '|{\|\|}|)
  (list "<<" 'oangle)
  (list ">>" 'cangle)
  (list "'" '|'|)
  (list "`" 'backquote))))
```

———-

6.0.169 defvar $infgeneric

— **postvars** —

```
(eval-when (eval load)
(prog ()
  (return
    ((lambda (var value)
       (loop
         (cond
           ((or (atom var) (progn (setq value (car var)) nil))
            (return nil))
           (t
            (setf (get (car value) 'infgeneric) (cadr value))))
         (setq var (cdr var))))
     (list
      (list 'equal '=)
      (list 'times '*)
      (list 'has '|has|)
      (list 'case '|case|)
      (list 'rem '|rem|)
      (list 'mod '|mod|)
      (list 'quo '|quo|)
      (list 'slash '/)
```

```
      (list 'backslash '|\\|)
      (list 'slashslash '//)
      (list 'backslashbackslash '|\\\\|)
      (list 'slashbackslash '|/\\|)
      (list 'backslashslash '|\\/|)
      (list 'power '**)
      (list 'carat '^)
      (list 'plus '+)
      (list 'minus '-)
      (list 'lt '<)
      (list 'gt '>)
      (list 'oangle '<<)
      (list 'cangle '>>)
      (list 'le '<=)
      (list 'ge '>=)
      (list 'notequal '~=)
      (list 'by '|by|)
      (list 'arrow '->)
      (list 'larrow '<-)
      (list 'bar '|\||)
      (list 'seg '|..|))
     nil))))
```

6.0.170 defun lineoftoks

lineoftoks bites off a token-dq from a line-stream returning the token-dq and the rest of the line-stream

```
;lineoftoks(s)==
;   $f: local:=nil
;   $r:local :=nil
;   $ln:local :=nil
;   $linepos:local:=nil
;   $n:local:=nil
;   $sz:local := nil
;   $floatok:local:=true
;   if not nextline s
;   then CONS(nil,nil)
;   else
;     if null scanIgnoreLine($ln,$n) -- line of spaces or starts ) or >
;     then cons(nil,$r)
;     else
;      toks:=[]
;      a:= incPrefix?('"command",1,$ln)
;      a =>
;                  $ln:=SUBSTRING($ln,8,nil)
```

```
;                b:= dqUnit constoken($ln,$linepos,["command",$ln],0)
;                cons([ [b,s] ],$r)
;
;      while $n<$sz repeat toks:=dqAppend(toks,scanToken())
;      if null toks
;      then cons([],$r)
;      else cons([ [toks,s] ],$r)
```

[nextline p110]
[scanIgnoreLine p110]
[incPrefix? p100]
[substring p??]
[dqUnit p341]
[constoken p111]
[$floatok p??]
[$f p??]
[$sz p??]
[$linepos p??]
[$r p??]
[$n p??]
[$ln p??]

— defun lineoftoks —

```
(defun |lineoftoks| (s)
 (let (|$floatok| |$sz| |$n| |$linepos| |$ln| |$r| |$f| |b| |a| |toks|)
 (declare (special |$floatok| |$f| |$sz| |$linepos| |$r| |$n| |$ln|))
  (setq |$f| nil)
  (setq |$r| nil)
  (setq |$ln| nil)
  (setq |$linepos| nil)
  (setq |$n| nil)
  (setq |$sz| nil)
  (setq |$floatok| t)
  (cond
   ((null (|nextline| s)) (cons nil nil))
   ((null (|scanIgnoreLine| |$ln| |$n|)) (cons nil |$r|))
   (t
    (setq |toks| nil)
    (setq |a| (|incPrefix?| "command" 1 |$ln|))
    (cond
     (|a|
      (setq |$ln| (substring |$ln| 8 nil))
      (setq |b|
       (|dqUnit| (|constoken| |$ln| |$linepos| (list '|command| |$ln|) 0)))
      (cons (list (list |b| s)) |$r|))
     (t
      ((lambda ()
        (loop
```

```
        (cond
         ((not (< |$n| |$sz|)) (return nil))
         (t (setq |toks| (|dqAppend| |toks| (|scanToken|))))))))
      (cond
       ((null |toks|) (cons nil |$r|))
       (t (cons (list (list |toks| s)) |$r|))))))))))
```

———-

6.0.171 defun nextline

[npNull p331]
[strposl p1040]
[$sz p??]
[$n p??]
[$linepos p??]
[$ln p??]
[$r p??]
[$f p??]

— **defun nextline** —

```
(defun |nextline| (s)
 (declare (special |$sz| |$n| |$linepos| |$ln| |$r| |$f|))
 (cond
  ((|npNull| s) nil)
  (t
   (setq |$f| (car s))
   (setq |$r| (cdr s))
   (setq |$ln| (cdr |$f|))
   (setq |$linepos| (caar |$f|))
   (setq |$n| (strposl " " |$ln| 0 t)) ; spaces at beginning
   (setq |$sz| (length |$ln|))
   t)))
```

———-

6.0.172 defun scanIgnoreLine

[qenum p1040]
[incPrefix? p100]

— **defun scanIgnoreLine** —

```
(defun |scanIgnoreLine| (ln n)
```

```
(cond
 ((null n) n)
 (t
  (cond
   ((char= (char ln 0) #\))
    (cond
      ((|incPrefix?| "command" 1 ln) t)
      (t nil)))
   (t n)))))
```

———-

6.0.173 defun constoken

[ncPutQ p414]

— **defun constoken** —

```
(defun |constoken| (ln lp b n)
 (declare (ignore ln))
 (let (a)
  (setq a (cons (elt b 0) (elt b 1)))
  (|ncPutQ| a '|posn| (cons lp n))
  a))
```

———-

6.0.174 defun scanToken

[qenum p1040]
[startsComment? p113]
[scanComment p113]
[startsNegComment? p114]
[scanNegComment p114]
[lfid p112]
[punctuation? p115]
[scanPunct p115]
[startsId? p1038]
[scanWord p123]
[scanSpace p127]
[scanString p127]
[digit? p119]
[scanNumber p129]
[scanEscape p132]

[scanError p132]
[dqUnit p341]
[constoken p111]
[lnExtraBlanks p343]
[$linepos p??]
[$n p??]
[$ln p??]

— **defun scanToken** —

```
(defun |scanToken| ()
 (let (b ch n linepos c ln)
 (declare (special |$linepos| |$n| |$ln|))
  (setq ln |$ln|)
  (setq c (qenum |$ln| |$n|))
  (setq linepos |$linepos|)
  (setq n |$n|)
  (setq ch (elt |$ln| |$n|))
  (setq b
   (cond
    ((|startsComment?|) (|scanComment|) nil)
    ((|startsNegComment?|) (|scanNegComment|) nil)
    ((char= c #\?)
      (setq |$n| (+ |$n| 1))
      (|lfid| "?"))
    ((|punctuation?| c) (|scanPunct|))
    ((|startsId?| ch) (|scanWord| nil))
    ((char= c #\space) (|scanSpace|) nil)
    ((char= c #\") (|scanString|))
    ((|digit?| ch) (|scanNumber|))
    ((char= c #\_) (|scanEscape|))
    (t (|scanError|))))
  (cond
    ((null b) nil)
    (t
     (|dqUnit|
      (|constoken| ln linepos b (+ n (|lnExtraBlanks| linepos))))))))
```

————-

6.0.175 defun lfid

To pair badge and badgee

— **defun lfid 0** —

```
(defun |lfid| (x)
 (list '|id| (intern x "BOOT")))
```

———-

6.0.176 defun startsComment?

[qenum p1040]
[$ln p??]
[$sz p??]
[$n p??]

— **defun startsComment?** —

```
(defun |startsComment?| ()
 (let (www)
 (declare (special |$ln| |$sz| |$n|))
  (cond
   ((< |$n| |$sz|)
    (cond
     ((char= (qenum |$ln| |$n|) #\+)
       (setq www (+ |$n| 1))
       (cond
        ((not (< www |$sz|)) nil)
        (t (char= (qenum |$ln| www) #\+))))
     (t nil)))
   (t nil))))
```

———-

6.0.177 defun scanComment

[lfcomment p114]
[substring p??]
[$ln p??]
[$sz p??]
[$n p??]

— **defun scanComment** —

```
(defun |scanComment| ()
 (let (n)
 (declare (special |$ln| |$sz| |$n|))
  (setq n |$n|)
  (setq |$n| |$sz|)
  (|lfcomment| (substring |$ln| n nil))))
```

———-

6.0.178 defun lfcomment

— **defun lfcomment 0** —

```
(defun |lfcomment| (x)
 (list '|comment| x))
```

———-

6.0.179 defun startsNegComment?

[qenum p1040]
[$ln p??]
[$sz p??]
[$n p??]

— **defun startsNegComment?** —

```
(defun |startsNegComment?| ()
 (let (www)
 (declare (special |$ln| |$sz| |$n|))
  (cond
   ((< |$n| |$sz|)
    (cond
     ((char= (qenum |$ln| |$n|) #\-)
       (setq www (+ |$n| 1))
       (cond
        ((not (< www |$sz|)) nil)
        (t (char= (qenum |$ln| www) #\-))))
     (t nil)))
   (t nil))))
```

———-

6.0.180 defun scanNegComment

[lfnegcomment p115]
[substring p??]
[$ln p??]
[$sz p??]
[$n p??]

— **defun scanNegComment** —

```
(defun |scanNegComment| ()
 (let (n)
 (declare (special |$ln| |$sz| |$n|))
  (setq n |$n|)
  (setq |$n| |$sz|)
  (|lfnegcomment| (substring |$ln| n nil))))
```

————-

6.0.181 defun lfnegcomment

— **defun lfnegcomment 0** —

```
(defun |lfnegcomment| (x)
 (list '|negcomment| x))
```

————-

6.0.182 defun punctuation?

— **defun punctuation?** —

```
(defun |punctuation?| (c)
 (eql (elt |scanPun| c) 1))
```

————-

6.0.183 defun scanPunct

[subMatch p116]
[scanError p132]
[scanKeyTr p117]
[$n p**??**]
[$ln p**??**]

— **defun scanPunct** —

```
(defun |scanPunct| ()
 (let (a sss)
 (declare (special |$n| |$ln|))
  (setq sss (|subMatch| |$ln| |$n|))
  (setq a (length sss))
  (cond
   ((eql a 0) (|scanError|))
   (t (setq |$n| (+ |$n| a)) (|scanKeyTr| sss)))))
```

———-

6.0.184 defun subMatch

[substringMatch p116]

— **defun subMatch** —

```
(defun |subMatch| (a b)
 (|substringMatch| a |scanDict| b))
```

———-

6.0.185 defun substringMatch

```
;substringMatch (l,d,i)==
;       h:= QENUM(l, i)
;       u:=ELT(d,h)
;       ll:=SIZE l
;       done:=false
;       s1:=’""
;       for j in 0.. SIZE u - 1 while not done repeat
;          s:=ELT(u,j)
;          ls:=SIZE s
;          done:=if ls+i > ll
;                then false
;                else
;                 eql:= true
;                 for k in 1..ls-1 while eql repeat
;                    eql:= EQL(QENUM(s,k),QENUM(l,k+i))
;                 if eql
;                 then
;                   s1:=s
;                   true
;                 else false
;       s1
```

[qenum p1040]
[size p1039]

— **defun substringMatch** —

```
(defun |substringMatch| (l dict i)
 (let (equl ls s s1 done ll u h)
  (setq h (qenum l i))
  (setq u (elt dict h))
  (setq ll (size l))
  (setq s1 "")
  ((lambda (Var4 j)
    (loop
     (cond
      ((or (> j Var4) done) (return nil))
      (t
       (setq s (elt u j))
       (setq ls (size s))
       (setq done
        (cond
         ((< ll (+ ls i)) nil)
         (t
          (setq equl t)
          ((lambda (Var5 k)
           (loop
            (cond
             ((or (> k Var5) (not equl)) (return nil))
             (t
              (setq equl (eql (qenum s k) (qenum l (+ k i))))))
             (setq k (+ k 1))))
           (- ls 1) 1)
          (cond (equl (setq s1 s) t) (t nil)))))))
     (setq j (+ j 1))))
   (- (size u) 1) 0)
  s1))
```

———-

6.0.186 defun scanKeyTr

[keyword p118]
[scanPossFloat p118]
[lfkey p119]
[scanCloser? p123]
[$floatok p**??**]

— **defun scanKeyTr** —

```
(defun |scanKeyTr| (w)
 (declare (special |$floatok|))
 (cond
  ((eq (|keyword| w) 'dot)
    (cond
     (|$floatok| (|scanPossFloat| w))
     (t (|lfkey| w))))
  (t (setq |$floatok| (null (|scanCloser?| w))) (|lfkey| w))))
```

———-

6.0.187 defun keyword

[hget p1038]

— **defun keyword 0** —

```
(defun |keyword| (st)
 (hget |scanKeyTable| st))
```

———-

6.0.188 defun keyword?

[hget p1038]

— **defun keyword? 0** —

```
(defun |keyword?| (st)
 (null (null (hget |scanKeyTable| st))))
```

———-

6.0.189 defun scanPossFloat

[digit? p119]
[lfkey p119]
[spleI p119]
[scanExponent p124]
[$ln p**??**]
[$sz p**??**]
[$n p**??**]

— **defun scanPossFloat** —

```
(defun |scanPossFloat| (w)
 (declare (special |$ln| |$sz| |$n|))
 (cond
  ((or (not (< |$n| |$sz|)) (null (|digit?| (elt |$ln| |$n|))))
    (|lfkey| w))
  (t
    (setq w (|spleI| #'|digit?|)) (|scanExponent| "0" w))))
```

———-

6.0.190 defun digit?

[digitp p1039]

— **defun digit?** —

```
(defun |digit?| (x)
 (digitp x))
```

———-

6.0.191 defun lfkey

[keyword p118]

— **defun lfkey** —

```
(defun |lfkey| (x)
 (list '|key| (|keyword| x)))
```

———-

6.0.192 defun spleI

[spleI1 p120]

— **defun spleI** —

```
(defun |spleI| (dig)
 (|spleI1| dig nil))
```

———-

6.0.193 defun spleI1

[qenum p1040]
[substring p??]
[scanEsc p120]
[spleI1 p120]
[concat p1041]
[$ln p??]
[$sz p??]
[$n p??]

— **defun spleI1** —

```
(defun |spleI1| (dig zro)
 (let (bb a str l n)
 (declare (special |$ln| |$sz| |$n|))
  (setq n |$n|)
  (setq l |$sz|)
  ; while $n<l and FUNCALL(dig,($ln.$n)) repeat $n:=$n+1
  ((lambda ()
    (loop
     (cond
      ((not (and (< |$n| l) (funcall dig (elt |$ln| |$n|))))
       (return nil))
      (t
       (setq |$n| (+ |$n| 1)))))))
  (cond
   ((or (equal |$n| l) (not (char= (qenum |$ln| |$n|) #\_)))
    (cond
      ((and (equal n |$n|) zro) "0")
      (t (substring |$ln| n (- |$n| n)))))
   (t                              ; escaped
    (setq str (substring |$ln| n (- |$n| n)))
    (setq |$n| (+ |$n| 1))
    (setq a (|scanEsc|))
    (setq bb (|spleI1| dig zro)) ; escape, any number of spaces are ignored
    (concat str bb)))))
```

———-

6.0.194 defun scanEsc

```
;scanEsc()==
;     if $n>=$sz
```

```
;     then if nextline($r)
;          then
;             while null $n repeat nextline($r)
;             scanEsc()
;             false
;          else false
;     else
;           n1:=STRPOSL('" ",$ln,$n,true)
;           if null n1
;           then if nextline($r)
;                then
;                  while null $n repeat nextline($r)
;                  scanEsc()
;                  false
;                else false
;           else
;                if $n=n1
;                then true
;                else if QENUM($ln,n1)=ESCAPE
;                     then
;                       $n:=n1+1
;                       scanEsc()
;                       false
;                     else
;                       $n:=n1
;                       startsNegComment?() or startsComment?() =>
;                                 nextline($r)
;                                 scanEsc()
;                                 false
;                       false
```

[nextline p110]
[scanEsc p120]
[strposl p1040]
[qenum p1040]
[startsNegComment? p114]
[startsComment? p113]
[$ln p**??**]
[$r p**??**]
[$sz p**??**]
[$n p**??**]

— **defun scanEsc** —

```
(defun |scanEsc| ()
 (let (n1)
 (declare (special |$ln| |$r| |$sz| |$n|))
  (cond
   ((not (< |$n| |$sz|))
```

```
    (cond
     ((|nextline| |$r|)
      ((lambda ()
       (loop
        (cond
         (|$n| (return nil))
         (t (|nextline| |$r|))))))
      (|scanEsc|)
      nil)
     (t nil)))
   (t
    (setq n1 (strposl " " |$ln| |$n| t))
    (cond
     ((null n1)
      (cond
       ((|nextline| |$r|)
        ((lambda ()
          (loop
           (cond
            (|$n| (return nil))
            (t (|nextline| |$r|))))))
         (|scanEsc|)
         nil)
        (t nil)))
     ((equal |$n| n1) t)
     ((char= (qenum |$ln| n1) #\_)
       (setq |$n| (+ n1 1))
       (|scanEsc|)
       nil)
     (t (setq |$n| n1)
      (cond
       ((or (|startsNegComment?|) (|startsComment?|))
        (progn
         (|nextline| |$r|)
         (|scanEsc|)
         nil))
       (t nil))))))))
```

———-

6.0.195 defvar $scanCloser

— **postvars** —

```
(eval-when (eval load)
  (defvar |scanCloser| (list ')| '} '] '|\|)| '|\|}| '|\|]|)))
```

———-

6.0.196 defun scanCloser?

[keyword p118]
[scanCloser p122]

— **defun scanCloser? 0** —

```
(defun |scanCloser?| (w)
 (declare (special |scanCloser|))
 (member (|keyword| w) |scanCloser|))
```

———-

6.0.197 defun scanWord

[scanW p125]
[lfid p112]
[keyword? p118]
[lfkey p119]
[$floatok p**??**]

— **defun scanWord** —

```
(defun |scanWord| (esp)
 (let (w aaa)
 (declare (special |$floatok|))
  (setq aaa (|scanW| nil))
  (setq w (elt aaa 1))
  (setq |$floatok| nil)
  (cond
   ((or esp (elt aaa 0))
    (|lfid| w))
   ((|keyword?| w)
    (setq |$floatok| t)
    (|lfkey| w))
   (t
    (|lfid| w)))))
```

———-

6.0.198 defun scanExponent

[lffloat p125]
[qenum p1040]
[digit? p119]
[spleI p119]
[concat p1041]
[$ln p??]
[$sz p??]
[$n p??]

— **defun scanExponent** —

```
(defun |scanExponent| (a w)
 (let (c1 e c n)
 (declare (special |$ln| |$sz| |$n|))
  (cond
   ((not (< |$n| |$sz|)) (|lffloat| a w "0"))
   (t
    (setq n |$n|)
    (setq c (qenum |$ln| |$n|))
    (cond
     ((or (char= c #\E) (char= c #\e))
      (setq |$n| (+ |$n| 1))
      (cond
       ((not (< |$n| |$sz|))
        (setq |$n| n)
        (|lffloat| a w "0"))
       ((|digit?| (elt |$ln| |$n|))
        (setq e (|spleI| #'|digit?|))
        (|lffloat| a w e))
       (t
        (setq c1 (qenum |$ln| |$n|))
        (cond
         ((or (char= c1 #\+) (char= c1 #\-))
           (setq |$n| (+ |$n| 1))
           (cond
            ((not (< |$n| |$sz|))
             (setq |$n| n)
             (|lffloat| a w "0"))
            ((|digit?| (elt |$ln| |$n|))
             (setq e (|spleI| #'|digit?|))
             (|lffloat| a w
              (cond
               ((char= c1 #\-)
                (concat "-" e))
               (t e))))
            (t
             (setq |$n| n)
```

```
          (|lffloat| a w "0")))))))))
    (t (|lffloat| a w "0")))))))
```

———-

6.0.199 defun lffloat

[concat p1041]

— **defun lffloat 0** —

```
(defun |lffloat| (a w e)
 (list '|float| (concat a "." w "e" e)))
```

———-

6.0.200 defmacro idChar?

— **defmacro idChar? 0** —

```
(defmacro |idChar?| (x)
 `(or (alphanumericp ,x) (member ,x '(#\? #\% #\' #\!) :test #'char=)))
```

———-

6.0.201 defun scanW

[posend p126]
[qenum p1040]
[substring p??]
[scanEsc p120]
[scanW p125]
[idChar? p125]
[concat p1041]
[$ln p??]
[$sz p??]
[$n p??]

— **defun scanW** —

```
(defun |scanW| (b)
```

```
(let (bb a str endid l n1)
(declare (special |$ln| |$sz| |$n|))
 (setq n1 |$n|)
 (setq |$n| (+ |$n| 1))
 (setq l |$sz|)
 (setq endid (|posend| |$ln| |$n|))
 (cond
  ((or (equal endid l) (not (char= (qenum |$ln| endid) #\_)))
   (setq |$n| endid)
   (list b (substring |$ln| n1 (- endid n1))))
  (t
   (setq str (substring |$ln| n1 (- endid n1)))
   (setq |$n| (+ endid 1))
   (setq a (|scanEsc|))
   (setq bb
    (cond
     (a (|scanW| t))
     ((not (< |$n| |$sz|)) (list b ""))
     ((|idChar?| (elt |$ln| |$n|)) (|scanW| b))
     (t (list b ""))))
   (list (or (elt bb 0) b) (concat str (elt bb 1)))))))
```

———-

6.0.202 defun posend

```
;posend(line,n)==
;     while n<#line and idChar? line.n repeat n:=n+1
;     n
```

NOTE: do not replace “lyne” with “line”

— **defun posend** —

```
(defun |posend| (lyne n)
 ((lambda ()
  (loop
   (cond
    ((not (and (< n (length lyne)) (|idChar?| (elt lyne n))))
     (return nil))
    (t (setq n (+ n 1)))))))
 n)
```

———-

6.0.203 defun scanSpace

[strposl p1040]
[lfspaces p127]
[$floatok p??]
[$ln p??]
[$n p??]

— **defun scanSpace** —

```
(defun |scanSpace| ()
 (let (n)
 (declare (special |$floatok| |$ln| |$n|))
  (setq n |$n|)
  (setq |$n| (strposl " " |$ln| |$n| t))
  (when (null |$n|) (setq |$n| (length |$ln|)))
  (setq |$floatok| t)
  (|lfspaces| (- |$n| n))))
```

————-

6.0.204 defun lfspaces

— **defun lfspaces 0** —

```
(defun |lfspaces| (x)
 (list '|spaces| x))
```

————-

6.0.205 defun scanString

[lfstring p128]
[scanS p128]
[$floatok p??]
[$n p??]

— **defun scanString** —

```
(defun |scanString| ()
 (declare (special |$floatok| |$n|))
  (setq |$n| (+ |$n| 1))
  (setq |$floatok| nil)
```

```
  (|lfstring| (|scanS|)))
```

———-

6.0.206 defun lfstring

— **defun lfstring 0** —

```
(defun |lfstring| (x)
 (if (eql (length x) 1)
  (list '|char| x)
  (list '|string| x)))
```

———-

6.0.207 defun scanS

[ncSoftError p349]
[lnExtraBlanks p343]
[strpos p1039]
[substring p??]
[scanEsc p120]
[concat p1041]
[scanTransform p129]
[scanS p128]
[$ln p??]
[$linepos p??]
[$sz p??]
[$n p??]

— **defun scanS** —

```
(defun |scanS| ()
 (let (b a str mn escsym strsym n)
 (declare (special |$ln| |$linepos| |$sz| |$n|))
 (cond
  ((not (< |$n| |$sz|))
   (|ncSoftError|
    (cons |$linepos| (+ (|lnExtraBlanks| |$linepos|) |$n|)) 'S2CN0001 nil) "")
  (t
   (setq n |$n|)
   (setq strsym (or (strpos "\"" |$ln| |$n| nil) |$sz|))
   (setq escsym (or (strpos "_" |$ln| |$n| nil) |$sz|))
```

```
(setq mn (min strsym escsym))
(cond
 ((equal mn |$sz|)
  (setq |$n| |$sz|)
  (|ncSoftError|
   (cons |$linepos| (+ (|lnExtraBlanks| |$linepos|) |$n|)) 'S2CN0001 nil)
    (substring |$ln| n nil))
 ((equal mn strsym)
  (setq |$n| (+ mn 1))
  (substring |$ln| n (- mn n)))
 (t
  (setq str (substring |$ln| n (- mn n)))
  (setq |$n| (+ mn 1))
  (setq a (|scanEsc|))
  (setq b
   (cond
    (a
     (setq str (concat str (|scanTransform| (elt |$ln| |$n|))))
     (setq |$n| (+ |$n| 1)) (|scanS|))
    (t (|scanS|))))
  (concat str b)))))))
```

———-

6.0.208 defun scanTransform

— **defun scanTransform** —

```
(defun |scanTransform| (x) x)
```

———-

6.0.209 defun scanNumber

[spleI p119]
[lfinteger p131]
[qenum p1040]
[spleI1 p120]
[scanExponent p124]
[scanCheckRadix p131]
[lfrinteger p131]
[concat p1041]
[$floatok p??]
[$ln p??]

[$sz p??]
[$n p??]

— defun scanNumber —

```
(defun |scanNumber| ()
 (let (v w n a)
 (declare (special |$floatok| |$ln| |$sz| |$n|))
  (setq a (|spleI| #'|digit?|))
  (cond
   ((not (< |$n| |$sz|))
    (|lfinteger| a))
   ((not (char= (qenum |$ln| |$n|) #\r))
    (cond
     ((and |$floatok| (char= (qenum |$ln| |$n|) #\.))
      (setq n |$n|)
      (setq |$n| (+ |$n| 1))
      (cond
       ((and (< |$n| |$sz|) (char= (qenum |$ln| |$n|) #\.))
        (setq |$n| n)
        (|lfinteger| a))
       (t
        (setq w (|spleI1| #'|digit?| t))
        (|scanExponent| a w))))
     (t (|lfinteger| a))))
   (t
    (setq |$n| (+ |$n| 1))
    (setq w (|spleI1| #'|rdigit?| t))
    (|scanCheckRadix| (parse-integer a) w)
    (cond
     ((not (< |$n| |$sz|))
      (|lfrinteger| a w))
     ((char= (qenum |$ln| |$n|) #\.)
      (setq n |$n|)
      (setq |$n| (+ |$n| 1))
      (cond
       ((and (< |$n| |$sz|) (char= (qenum |$ln| |$n|) #\.))
        (setq |$n| n)
        (|lfrinteger| a w))
       (t
        (setq v (|spleI1| #'|rdigit?| t))
        (|scanCheckRadix| (parse-integer a) v)
        (|scanExponent| (concat a "r" w) v))))
     (t (|lfrinteger| a w)))))))
```

———-

6.0.210 defun rdigit?

[strpos p1039]

— **defun rdigit? 0** —

```
(defun |rdigit?| (x)
 (strpos x "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" 0 nil))
```

———-

6.0.211 defun lfinteger

— **defun lfinteger 0** —

```
(defun |lfinteger| (x)
 (list '|integer| x))
```

———-

6.0.212 defun lfrinteger

[concat p1041]

— **defun lfrinteger 0** —

```
(defun |lfrinteger| (r x)
 (list '|integer| (concat r (concat "r" x))))
```

———-

6.0.213 defun scanCheckRadix

```
;scanCheckRadix(r,w)==
;       ns:=#w
;       done:=false
;       for i in 0..ns-1  repeat
;         a:=rdigit? w.i
;         if null a or a>=r
;         then  ncSoftError(cons($linepos,lnExtraBlanks $linepos+$n-ns+i),
;                    "S2CN0002", [w.i])
```

[$n p??]
[$linepos p??]

— **defun scanCheckRadix** —

```
(defun |scanCheckRadix| (r w)
 (let (a ns)
 (declare (special |$n| |$linepos|))
  (setq ns (length w))
  ((lambda (Var1 i)
   (loop
    (cond
     ((> i Var1) (return nil))
     (t
      (setq a (|rdigit?| (elt w i)))
      (cond
       ((or (null a) (not (< a r)))
        (|ncSoftError|
         (cons |$linepos| (+ (- (+ (|lnExtraBlanks| |$linepos|) |$n|) ns) i))
          'S2CN0002 (list (elt w i)))))))
    (setq i (+ i 1))))
   (- ns 1) 0)))
```

————-

6.0.214 defun scanEscape

[scanEsc p120]
[scanWord p123]
[$n p??]

— **defun scanEscape** —

```
(defun |scanEscape| ()
 (declare (special |$n|))
  (setq |$n| (+ |$n| 1))
  (when (|scanEsc|) (|scanWord| t)))
```

————-

6.0.215 defun scanError

[ncSoftError p349]
[lnExtraBlanks p343]
[lferror p133]

[$ln p??]
[$linepos p??]
[$n p??]

— **defun scanError** —

```
(defun |scanError| ()
 (let (n)
 (declare (special |$ln| |$linepos| |$n|))
  (setq n |$n|)
  (setq |$n| (+ |$n| 1))
  (|ncSoftError|
   (cons |$linepos| (+ (|lnExtraBlanks| |$linepos|) |$n|))
    'S2CN0003 (list (elt |$ln| n)))
  (|lferror| (elt |$ln| n))))
```

———-

6.0.216 defun lferror

— **defun lferror 0** —

```
(defun |lferror| (x)
 (list '|error| x))
```

———-

6.0.217 defvar $scanKeyTable

— **postvars** —

```
(eval-when (eval load)
 (defvar |scanKeyTable| (|scanKeyTableCons|)))
```

———-

6.0.218 defun scanKeyTableCons

This function is used to build the scanKeyTable

```
;scanKeyTableCons()==
;   KeyTable:=MAKE_-HASHTABLE("CVEC",true)
;   for st in scanKeyWords repeat
;      HPUT(KeyTable,CAR st,CADR st)
;   KeyTable
```

— **defun scanKeyTableCons** —

```
(defun |scanKeyTableCons| ()
 (let (KeyTable)
  (setq KeyTable (make-hash-table :test #'equal))
  ((lambda (Var6 st)
   (loop
    (cond
     ((or (atom Var6) (progn (setq st (car Var6)) nil))
      (return nil))
     (t
      (hput KeyTable (car st) (cadr st))))
    (setq Var6 (cdr Var6))))
   |scanKeyWords| nil)
  KeyTable))
```

———-

6.0.219 defvar $scanDict

— **postvars** —

```
(eval-when (eval load)
 (defvar |scanDict| (|scanDictCons|)))
```

———-

6.0.220 defun scanDictCons

```
;scanDictCons()==
;      l:= HKEYS scanKeyTable
;      d :=
;          a:=MAKE_-VEC(256)
;          b:=MAKE_-VEC(1)
;          VEC_-SETELT(b,0,MAKE_-CVEC 0)
;          for i in 0..255 repeat VEC_-SETELT(a,i,b)
;          a
```

```
;      for s in l repeat scanInsert(s,d)
;      d
```

[hkeys p1038]

— defun scanDictCons —

```
(defun |scanDictCons| ()
 (let (d b a l)
  (setq l (hkeys |scanKeyTable|))
  (setq d
   (progn
    (setq a (make-array 256))
    (setq b (make-array 1))
    (setf (svref b 0)
      (make-array 0 :fill-pointer 0 :element-type 'string-char))
    ((lambda (i)
     (loop
      (cond
       ((> i 255) (return nil))
       (t (setf (svref a i) b)))
      (setq i (+ i 1))))
     0)
    a))
  ((lambda (Var7 s)
   (loop
    (cond
     ((or (atom Var7) (progn (setq s (car Var7)) nil))
      (return nil))
     (t (|scanInsert| s d)))
    (setq Var7 (cdr Var7))))
   l nil)
  d))
```

———-

6.0.221 defun scanInsert

```
;scanInsert(s,d) ==
;      l := #s
;      h := QENUM(s,0)
;      u := ELT(d,h)
;      n := #u
;      k:=0
;      while l <= #(ELT(u,k)) repeat
;          k:=k+1
;      v := MAKE_-VEC(n+1)
```

```
;      for i in 0..k-1 repeat VEC_-SETELT(v,i,ELT(u,i))
;      VEC_-SETELT(v,k,s)
;      for i in k..n-1 repeat VEC_-SETELT(v,i+1,ELT(u,i))
;      VEC_-SETELT(d,h,v)
;      s
```

[qenum p1040]

— defun scanInsert —

```
(defun |scanInsert| (s d)
 (let (v k n u h l)
  (setq l (length s))
  (setq h (qenum s 0))
  (setq u (elt d h))
  (setq n (length u))
  (setq k 0)
  ((lambda ()
   (loop
    (cond
     ((< (length (elt u k)) l) (return nil))
     (t (setq k (+ k 1)))))))
  (setq v (make-array (+ n 1)))
  ((lambda (Var2 i)
   (loop
    (cond
     ((> i Var2) (return nil))
     (t (setf (svref v i) (elt u i))))
    (setq i (+ i 1))))
   (- k 1) 0)
  (setf (svref v k) s)
  ((lambda (Var3 i)
   (loop
    (cond
     ((> i Var3) (return nil))
     (t (setf (svref v (+ i 1)) (elt u i))))
    (setq i (+ i 1))))
   (- n 1) k)
  (setf (svref d h) v)
  s))
```

———-

6.0.222 defvar $scanPun

— postvars —

```
(eval-when (eval load)
 (defvar |scanPun| (|scanPunCons|)))
```

———-

6.0.223 defun scanPunCons

```
;scanPunCons()==
;    listing := HKEYS scanKeyTable
;    a:=MAKE_-BVEC 256
;    for i in 0..255 repeat BVEC_-SETELT(a,i,0)
;    for k in listing repeat
;       if not startsId? k.0
;       then BVEC_-SETELT(a,QENUM(k,0),1)
;    a
```

[hkeys p1038]

— **defun scanPunCons** —

```
(defun |scanPunCons| ()
 (let (a listing)
  (setq listing (hkeys |scanKeyTable|))
  (setq a (make-array (list 256) :element-type 'bit :initial-element 0))
  ((lambda (i)
   (loop
    (cond
     ((> i 255) (return nil))
     (t (setf (sbit a i) 0)))
    (setq i (+ i 1))))
   0)
  ((lambda (Var8 k)
   (loop
    (cond
     ((or (atom Var8) (progn (setq k (car Var8)) nil))
      (return nil))
     (t
      (cond
       ((null (|startsId?| (elt k 0)))
        (setf (sbit a (qenum k 0)) 1)))))
    (setq Var8 (cdr Var8))))
   listing nil)
  a))
```

———-

Chapter 7

Input Stream Parser

7.0.224 defun Input Stream Parser

[trappoint p**??**]
[npFirstTok p141]
[npItem p140]
[ncSoftError p349]
[tokPosn p411]
[pfWrong p294]
[pfDocument p244]
[pfListOf p243]
[$ttok p**??**]
[$stok p**??**]
[$stack p**??**]
[$inputStream p**??**]

— defun npParse —

```
(defun |npParse| (stream)
 (let (|$ttok| |$stok| |$stack| |$inputStream| found)
 (declare (special |$ttok| |$stack| |$inputStream| |$stok|))
  (setq |$inputStream| stream)
  (setq |$stack| nil)
  (setq |$stok| nil)
  (setq |$ttok| nil)
  (|npFirstTok|)
  (setq found (catch 'trappoint (|npItem|)))
  (cond
   ((eq found 'trapped)
    (|ncSoftError| (|tokPosn| |$stok|) 's2cy0006 nil)
    (|pfWrong| (|pfDocument| "top level syntax error") (|pfListOf| nil)))
   ((null (null |$inputStream|))
```

```
    (|ncSoftError| (|tokPosn| |$stok|) 's2cy0002 nil)
    (|pfWrong|
     (|pfDocument| (list "input stream not exhausted"))
     (|pfListOf| nil)))
   ((null |$stack|)
    (|ncSoftError| (|tokPosn| |$stok|) 's2cy0009 nil)
    (|pfWrong| (|pfDocument| (list "stack empty")) (|pfListOf| nil)))
   (t (car |$stack|)))))
```

———-

7.0.225 defun npItem

[npQualDef p143]
[npEqKey p143]
[npItem1 p140]
[npPop1 p142]
[pfEnSequence p261]
[npPush p141]
[pfNovalue p277]

— **defun npItem** —

```
(defun |npItem| ()
 (let (c b a tmp1)
  (when (|npQualDef|)
    (if (|npEqKey| 'semicolon)
     (progn
      (setq tmp1 (|npItem1| (|npPop1|)))
      (setq a (car tmp1))
      (setq b (cadr tmp1))
      (setq c (|pfEnSequence| b))
      (if a
       (|npPush| c)
       (|npPush| (|pfNovalue| c))))
     (|npPush| (|pfEnSequence| (|npPop1|)))))))
```

———-

7.0.226 defun npItem1

[npQualDef p143]
[npEqKey p143]
[npItem1 p140]
[npPop1 p142]

— **defun npItem1** —

```
(defun |npItem1| (c)
 (let (b a tmp1)
  (if (|npQualDef|)
   (if (|npEqKey| 'semicolon)
     (progn
      (setq tmp1 (|npItem1| (|npPop1|)))
      (setq a (car tmp1))
      (setq b (cadr tmp1))
      (list a (append c b)))
    (list t (append c (|npPop1|))))
   (list nil c))))
```

———-

7.0.227 defun npFirstTok

Sets the current leaf ($stok) to the next leaf in the input stream. Sets the current token ($ttok) cdr of the leaf. A leaf token looks like [head, token, position] where head is either an id or (id . alist) [tokConstruct p409]
[tokPosn p411]
[tokPart p411]
[$ttok p**??**]
[$stok p**??**]
[$inputStream p**??**]

— **defun npFirstTok** —

```
(defun |npFirstTok| ()
 (declare (special |$ttok| |$stok| |$inputStream|))
 (if (null |$inputStream|)
  (setq |$stok| (|tokConstruct| 'error 'nomore (|tokPosn| |$stok|)))
  (setq |$stok| (car |$inputStream|)))
 (setq |$ttok| (|tokPart| |$stok|)))
```

———-

7.0.228 defun Push one item onto $stack

[$stack p**??**]

— **defun npPush 0** —

```
(defun |npPush| (x)
 (declare (special |$stack|))
  (push x |$stack|))
```

———-

7.0.229 defun Pop one item off $stack

[$stack p??]

— **defun npPop1 0** —

```
(defun |npPop1| ()
 (declare (special |$stack|))
 (pop |$stack|))
```

———-

7.0.230 defun Pop the second item off $stack

[$stack p??]

— **defun npPop2 0** —

```
(defun |npPop2| ()
 (let (a)
 (declare (special |$stack|))
  (setq a (cadr |$stack|))
  (rplacd |$stack| (cddr |$stack|))
  a))
```

———-

7.0.231 defun Pop the third item off $stack

[$stack p??]

— **defun npPop3 0** —

```
(defun |npPop3| ()
 (let (a)
 (declare (special |$stack|))
  (setq a (caddr |$stack|))
```

```
  (rplacd (cdr |$stack|) (cdddr |$stack|)) a))
```

———-

7.0.232 defun npQualDef

[npComma p144]
[npPush p141]
[npPop1 p142]

— **defun npQualDef** —

```
(defun |npQualDef| ()
 (and (|npComma|) (|npPush| (list (|npPop1|)))))
```

———-

7.0.233 defun Advance over a keyword

Test for the keyword, if found advance the token stream [npNext p143]
[$ttok p**??**]
[$stok p**??**]

— **defun npEqKey** —

```
(defun |npEqKey| (keyword)
 (declare (special |$ttok| |$stok|))
 (and
  (eq (caar |$stok|) '|key|)
  (eq keyword |$ttok|)
  (|npNext|)))
```

———-

7.0.234 defun Advance the input stream

This advances the input stream. The call to npFirstTok picks off the next token in the input stream and updates the current leaf ($stok) and the current token ($ttok) [npFirstTok p141]
[$inputStream p**??**]

— **defun npNext** —

```
(defun |npNext| ()
 (declare (special |$inputStream|))
  (setq |$inputStream| (cdr |$inputStream|))
  (|npFirstTok|))
```

———-

7.0.235 defun npComma

[npTuple p144]
[npQualifiedDefinition p145]

— **defun npComma** —

```
(defun |npComma| ()
 (|npTuple| #'|npQualifiedDefinition|))
```

———-

7.0.236 defun npTuple

[npListofFun p219]
[npCommaBackSet p144]
[pfTupleListOf p290]

— **defun npTuple** —

```
(defun |npTuple| (|p|)
 (|npListofFun| |p| #'|npCommaBackSet| #'|pfTupleListOf|))
```

———-

7.0.237 defun npCommaBackSet

[npEqKey p143]

— **defun npCommaBackSet** —

```
(defun |npCommaBackSet| ()
 (and
  (|npEqKey| 'comma)
  (or (|npEqKey| 'backset) t)))
```

———-

7.0.238 defun npQualifiedDefinition

[npQualified p145]
[npDefinitionOrStatement p145]

— **defun npQualifiedDefinition** —

```
(defun |npQualifiedDefinition| ()
 (|npQualified| #'|npDefinitionOrStatement|))
```

———-

7.0.239 defun npQualified

[npEqKey p143]
[npDefinition p165]
[npTrap p210]
[npPush p141]
[pfWhere p292]
[npPop1 p142]
[npLetQualified p164]

— **defun npQualified** —

```
(defun |npQualified| (f)
 (if (funcall f)
  (progn
   (do () ; while ... do
      ((not (and (|npEqKey| 'where) (or (|npDefinition|) (|npTrap|)))))
   (|npPush| (|pfWhere| (|npPop1|) (|npPop1|))))
   t)
  (|npLetQualified| f)))
```

———-

7.0.240 defun npDefinitionOrStatement

[npBackTrack p146]
[npGives p146]
[npDef p185]

— **defun npDefinitionOrStatement** —

```
(defun |npDefinitionOrStatement| ()
 (|npBackTrack| #'|npGives| 'def #'|npDef|))
```

———-

7.0.241 defun npBackTrack

[npState p210]
[npEqPeek p150]
[npRestore p150]
[npTrap p210]

— **defun npBackTrack** —

```
(defun |npBackTrack| (p1 p2 p3)
 (let (a)
  (setq a (|npState|))
  (when (apply p1 nil)
    (cond
     ((|npEqPeek| p2)
       (|npRestore| a)
       (or (apply p3 nil) (|npTrap|)))
     (t t)))))
```

———-

7.0.242 defun npGives

[npBackTrack p146]
[npExit p213]
[npLambda p146]

— **defun npGives** —

```
(defun |npGives| ()
 (|npBackTrack| #'|npExit| 'gives #'|npLambda|))
```

———-

7.0.243 defun npLambda

[npVariable p211]
[npLambda p146]

[npTrap p210]
[npPush p141]
[pfLam p270]
[npPop2 p142]
[npPop1 p142]
[npEqKey p143]
[npDefinitionOrStatement p145]
[npType p147]
[pfReturnTyped p283]

— **defun npLambda** —

```
(defun |npLambda| ()
 (or
  (and
   (|npVariable|)
   (or (|npLambda|) (|npTrap|))
   (|npPush| (|pfLam| (|npPop2|) (|npPop1|))))
  (and
   (|npEqKey| 'gives)
   (or (|npDefinitionOrStatement|) (|npTrap|)))
  (and
   (|npEqKey| 'colon)
   (or (|npType|) (|npTrap|))
   (|npEqKey| 'gives)
   (or (|npDefinitionOrStatement|) (|npTrap|))
   (|npPush| (|pfReturnTyped| (|npPop2|) (|npPop1|))))))
```

————-

7.0.244 defun npType

[npMatch p148]
[npPop1 p142]
[npWith p148]
[npPush p141]

— **defun npType** —

```
(defun |npType| ()
 (and
  (|npMatch|)
  (let ((a (|npPop1|)))
   (or
    (|npWith| a)
    (|npPush| a)))))
```

———-

7.0.245 defun npMatch

[npLeftAssoc p204]
[npSuch p148]

— **defun npMatch** —

```
(defun |npMatch| ()
 (|npLeftAssoc| '(is isnt) #'|npSuch|))
```

———-

7.0.246 defun npSuch

[npLeftAssoc p204]
[npLogical p195]

— **defun npSuch** —

```
(defun |npSuch| ()
 (|npLeftAssoc| '(bar) #'|npLogical|))
```

———-

7.0.247 defun npWith

[npEqKey p143]
[npState p210]
[npCategoryL p150]
[npTrap p210]
[npEqPeek p150]
[npRestore p150]
[npVariable p211]
[npCompMissing p149]
[npPush p141]
[pfWith p294]
[npPop2 p142]
[npPop1 p142]
[pfNothing p243]

— **defun npWith** —

```
(defun |npWith| (extra)
 (let (a)
  (and
   (|npEqKey| 'with)
   (progn
    (setq a (|npState|))
    (or (|npCategoryL|) (|npTrap|))
    (if (|npEqPeek| 'in)
     (progn
       (|npRestore| a)
       (and
        (or (|npVariable|) (|npTrap|))
        (|npCompMissing| 'in)
        (or (|npCategoryL|) (|npTrap|))
        (|npPush| (|pfWith| (|npPop2|) (|npPop1|) extra))))
     (|npPush| (|pfWith| (|pfNothing|) (|npPop1|) extra)))))))
```

———-

7.0.248 defun npCompMissing

[npEqKey p143]
[npMissing p149]

— **defun npCompMissing** —

```
(defun |npCompMissing| (s)
 (or (|npEqKey| s) (|npMissing| s)))
```

———-

7.0.249 defun npMissing

[trappoint p**??**]
[ncSoftError p349]
[tokPosn p411]
[pname p1039]
[$stok p**??**]

— **defun npMissing** —

```
(defun |npMissing| (s)
 (declare (special |$stok|))
  (|ncSoftError| (|tokPosn| |$stok|) 'S2CY0007 (list (pname s)))
  (throw 'trappoint 'trapped)))))
```

———-

7.0.250 defun npRestore

[npFirstTok p141]
[$stack p??]
[$inputStream p??]

— **defun npRestore** —

```
(defun |npRestore| (x)
 (declare (special |$stack| |$inputStream|))
  (setq |$inputStream| (car x))
  (|npFirstTok|)
  (setq |$stack| (cdr x))
  t)
```

———-

7.0.251 defun Peek for keyword s, no advance of token stream

[$ttok p??]
[$stok p??]

— **defun npEqPeek 0** —

```
(defun |npEqPeek| (s)
 (declare (special |$ttok| |$stok|))
  (and (eq (caar |$stok|) '|key|) (eq s |$ttok|)))
```

———-

7.0.252 defun npCategoryL

[npCategory p151]
[npPush p141]
[pfUnSequence p291]
[npPop1 p142]

— **defun npCategoryL** —

```
(defun |npCategoryL| ()
 (and
  (|npCategory|)
  (|npPush| (|pfUnSequence| (|npPop1|)))))
```

———-

7.0.253 defun npCategory

[npPP p207]
[npSCategory p151]

— **defun npCategory** —

```
(defun |npCategory| ()
 (|npPP| #'|npSCategory|))
```

———-

7.0.254 defun npSCategory

[npWConditional p193]
[npCategoryL p150]
[npPush p141]
[npPop1 p142]
[npDefaultValue p192]
[npState p210]
[npPrimary p155]
[npEqPeek p150]
[npRestore p150]
[npSignature p152]
[npApplication p160]
[pfAttribute p251]
[npTrap p210]

— **defun npSCategory** —

```
(defun |npSCategory| ()
 (let (a)
  (cond
   ((|npWConditional| #'|npCategoryL|) (|npPush| (list (|npPop1|))))
   ((|npDefaultValue|) t)
   (t
    (setq a (|npState|))
```

```
    (cond
     ((|npPrimary|)
       (cond
        ((|npEqPeek| 'colon) (|npRestore| a) (|npSignature|))
        (t
         (|npRestore| a)
         (or
          (and (|npApplication|) (|npPush| (list (|pfAttribute| (|npPop1|)))))
          (|npTrap|)))))
      (t nil))))
```

———-

7.0.255 defun npSignature

[npSigItemlist p152]
[npPush p141]
[pfWDec p291]
[pfNothing p243]
[npPop1 p142]

— **defun npSignature** —

```
(defun |npSignature| ()
 (and (|npSigItemlist|) (|npPush| (|pfWDec| (|pfNothing|) (|npPop1|)))))
```

———-

7.0.256 defun npSigItemlist

[npListing p153]
[npSigItem p154]
[npPush p141]
[pfListOf p243]
[pfAppend p253]
[pfParts p247]
[npPop1 p142]

— **defun npSigItemlist** —

```
(defun |npSigItemlist| ()
 (and
  (|npListing| #'|npSigItem|)
  (|npPush| (|pfListOf| (|pfAppend| (|pfParts| (|npPop1|)))))))
```

———-

7.0.257 defun npListing

[npList p153]
[pfListOf p243]

— **defun npListing** —

```
(defun |npListing| (p)
 (|npList| p 'comma #'|pfListOf|))
```

———-

7.0.258 defun Always produces a list, fn is applied to it

[npEqKey p143]
[npTrap p210]
[npPush p141]
[npPop3 p142]
[npPop2 p142]
[npPop1 p142]
[$stack p??]

— **defun npList** —

```
(defun |npList| (f str1 fn)
 (let (a)
 (declare (special |$stack|))
  (cond
   ((apply f nil)
    (cond
     ((and (|npEqKey| str1)
           (or (|npEqKey| 'backset) t)
           (or (apply f nil) (|npTrap|)))
       (setq a |$stack|)
       (setq |$stack| nil)
       (do ()                                ; while .. do nothing
           ((not
             (and (|npEqKey| str1)
                  (or (|npEqKey| 'backset) t)
                  (or (apply f nil) (|npTrap|))))
            nil))
       (setq |$stack| (cons (nreverse |$stack|) a))
       (|npPush| (funcall fn (cons (|npPop3|) (cons (|npPop2|) (|npPop1|))))))
     (t (|npPush| (funcall fn (list (|npPop1|))))))))
```

```
   (t (|npPush| (funcall fn nil))))))
```

———-

7.0.259 defun npSigItem

[npTypeVariable p154]
[npSigDecl p155]
[npTrap p210]

— **defun npSigItem** —

```
(defun |npSigItem| ()
 (and (|npTypeVariable|) (or (|npSigDecl|) (|npTrap|))))
```

———-

7.0.260 defun npTypeVariable

[npParenthesized p212]
[npTypeVariablelist p155]
[npSignatureDefinee p154]
[npPush p141]
[pfListOf p243]
[npPop1 p142]

— **defun npTypeVariable** —

```
(defun |npTypeVariable| ()
 (or
  (|npParenthesized| #'|npTypeVariablelist|)
  (and (|npSignatureDefinee|) (|npPush| (|pfListOf| (list (|npPop1|)))))))
```

———-

7.0.261 defun npSignatureDefinee

[npName p202]
[npInfixOperator p158]
[npPrefixColon p159]

— **defun npSignatureDefinee** —

```
(defun |npSignatureDefinee| ()
 (or (|npName|) (|npInfixOperator|) (|npPrefixColon|)))
```

———-

7.0.262 defun npTypeVariablelist

[npListing p153]
[npSignatureDefinee p154]

— **defun npTypeVariablelist** —

```
(defun |npTypeVariablelist| ()
 (|npListing| #'|npSignatureDefinee|))
```

———-

7.0.263 defun npSigDecl

[npEqKey p143]
[npType p147]
[npTrap p210]
[npPush p141]
[pfSpread p237]
[pfParts p247]
[npPop2 p142]
[npPop1 p142]

— **defun npSigDecl** —

```
(defun |npSigDecl| ()
 (and
  (|npEqKey| 'colon)
  (or (|npType|) (|npTrap|))
  (|npPush| (|pfSpread| (|pfParts| (|npPop2|)) (|npPop1|)))))
```

———-

7.0.264 defun npPrimary

[npPrimary1 p162]
[npPrimary2 p156]

— **defun npPrimary** —

```
(defun |npPrimary| ()
 (or (|npPrimary1|) (|npPrimary2|)))
```

————-

7.0.265 defun npPrimary2

[npEncAp p180]
[npAtom2 p157]
[npAdd p157]
[pfNothing p243]
[npWith p148]

— **defun npPrimary2** —

```
(defun |npPrimary2| ()
 (or
  (|npEncAp| #'|npAtom2|)
  (|npAdd| (|pfNothing|))
  (|npWith| (|pfNothing|))))
```

————-

7.0.266 defun npADD

TPDHERE: Note that there is also an npAdd function [npType p147]
[npPop1 p142]
[npAdd p157]
[npPush p141]

— **defun npADD** —

```
(defun |npADD| ()
 (let (a)
  (and
   (|npType|)
   (progn
    (setq a (|npPop1|))
    (or
      (|npAdd| a)
      (|npPush| a))))))
```

———-

7.0.267 defun npAdd

TPDHERE: Note that there is also an npADD function [npEqKey p143]
[npState p210]
[npDefinitionOrStatement p145]
[npTrap p210]
[npEqPeek p150]
[npRestore p150]
[npVariable p211]
[npCompMissing p149]
[npDefinitionOrStatement p145]
[npPush p141]
[pfAdd p250]
[npPop2 p142]
[npPop1 p142]
[pfNothing p243]

— **defun npAdd** —

```
(defun |npAdd| (extra)
 (let (a)
  (and
   (|npEqKey| 'add)
   (progn
    (setq a (|npState|))
    (or (|npDefinitionOrStatement|) (|npTrap|))
    (cond
     ((|npEqPeek| 'in)
      (progn
        (|npRestore| a)
        (and
         (or (|npVariable|) (|npTrap|))
         (|npCompMissing| 'in)
         (or (|npDefinitionOrStatement|) (|npTrap|))
         (|npPush| (|pfAdd| (|npPop2|) (|npPop1|) extra)))))
     (t
       (|npPush| (|pfAdd| (|pfNothing|) (|npPop1|) extra)))))))))
```

———-

7.0.268 defun npAtom2

[npInfixOperator p158]
[npAmpersand p202]

[npPrefixColon p159]
[npFromdom p200]

— **defun npAtom2** —

```
(defun |npAtom2| ()
 (and
  (or (|npInfixOperator|) (|npAmpersand|) (|npPrefixColon|))
  (|npFromdom|)))
```

————-

7.0.269 defun npInfixOperator

[npInfixOp p159]
[npState p210]
[npEqKey p143]
[npInfixOp p159]
[npPush p141]
[pfSymb p249]
[npPop1 p142]
[tokPosn p411]
[npRestore p150]
[tokConstruct p409]
[tokPart p411]
[$stok p**??**]

— **defun npInfixOperator** —

```
(defun |npInfixOperator| ()
 (let (b a)
  (declare (special |$stok|))
   (or (|npInfixOp|)
       (progn
        (setq a (|npState|))
        (setq b |$stok|)
        (cond
         ((and (|npEqKey| '|'|) (|npInfixOp|))
           (|npPush| (|pfSymb| (|npPop1|) (|tokPosn| b))))
          (t
           (|npRestore| a)
           (cond
            ((and (|npEqKey| 'backquote) (|npInfixOp|))
              (setq a (|npPop1|))
               (|npPush| (|tokConstruct| '|idsy| (|tokPart| a) (|tokPosn| a))))
            (t
```

```
 (|npRestore| a)
 nil)))))))))
```

———-

7.0.270 defun npInfixOp

[npPushId p207]
[$ttok p**??**]
[$stok p**??**]

— **defun npInfixOp** —

```
(defun |npInfixOp| ()
 (declare (special |$ttok| |$stok|))
  (and
   (eq (caar |$stok|) '|key|)
   (get |$ttok| 'infgeneric)
   (|npPushId|)))
```

———-

7.0.271 defun npPrefixColon

[npEqPeek p150]
[npPush p141]
[tokConstruct p409]
[tokPosn p411]
[npNext p143]
[$stok p**??**]

— **defun npPrefixColon** —

```
(defun |npPrefixColon| ()
 (declare (special |$stok|))
 (and
  (|npEqPeek| 'colon)
  (progn
   (|npPush| (|tokConstruct| '|id| '|:| (|tokPosn| |$stok|)))
   (|npNext|))))
```

———-

7.0.272 defun npApplication

[npDotted p160]
[npPrimary p155]
[npApplication2 p161]
[npPush p141]
[pfApplication p251]
[npPop2 p142]
[npPop1 p142]

— **defun npApplication** —

```
(defun |npApplication| ()
 (and
  (|npDotted| #'|npPrimary|)
  (or
   (and
    (|npApplication2|)
    (|npPush| (|pfApplication| (|npPop2|) (|npPop1|))))
    t)))
```

————-

7.0.273 defun npDotted

[p??]

— **defun npDotted** —

```
(defun |npDotted| (f)
 (and (apply f nil) (|npAnyNo| #'|npSelector|)))
```

————-

7.0.274 defun npAnyNo

fn must transform the head of the stack

— **defun npAnyNo 0** —

```
(defun |npAnyNo| (fn)
 (do () ((not (apply fn nil)))) ; while apply do...
 t)
```

————-

7.0.275 defun npSelector

[npEqKey p143]
[npPrimary p155]
[npTrap p210]
[npPush p141]
[pfApplication p251]
[npPop2 p142]
[npPop1 p142]

— **defun npSelector** —

```
(defun |npSelector| ()
 (and
  (|npEqKey| 'dot)
  (or (|npPrimary|) (|npTrap|))
  (|npPush| (|pfApplication| (|npPop2|) (|npPop1|)))))
```

———-

7.0.276 defun npApplication2

[npDotted p160]
[npPrimary1 p162]
[npApplication2 p161]
[npPush p141]
[pfApplication p251]
[npPop2 p142]
[npPop1 p142]

— **defun npApplication2** —

```
(defun |npApplication2| ()
 (and
  (|npDotted| #'|npPrimary1|)
  (or
   (and
    (|npApplication2|)
    (|npPush| (|pfApplication| (|npPop2|) (|npPop1|))))
    t)))
```

———-

7.0.277 defun npPrimary1

[npEncAp p180]
[npAtom1 p181]
[npLet p164]
[npFix p164]
[npMacro p162]
[npBPileDefinition p186]
[npDefn p185]
[npRule p191]

— **defun npPrimary1** —

```
(defun |npPrimary1| ()
 (or
  (|npEncAp| #'|npAtom1|)
  (|npLet|)
  (|npFix|)
  (|npMacro|)
  (|npBPileDefinition|)
  (|npDefn|)
  (|npRule|)))
```

———-

7.0.278 defun npMacro

[npPP p207]
[npMdef p162]

— **defun npMacro** —

```
(defun |npMacro| ()
 (and
  (|npEqKey| 'macro)
  (|npPP| #'|npMdef|)))
```

———-

7.0.279 defun npMdef

TPDHERE: Beware that this function occurs with uppercase also [npQuiver p196]
[pfCheckMacroOut p238]
[npPop1 p142]

[npDefTail p192]
[npTrap p210]
[npPop1 p142]
[npPush p141]
[pfMacro p275]
[pfPushMacroBody p241]

— **defun npMdef** —

```
(defun |npMdef| ()
 (let (body arg op tmp)
  (when (|npQuiver|)                    ;[op,arg]:= pfCheckMacroOut(npPop1())
   (setq tmp (|pfCheckMacroOut| (|npPop1|)))
   (setq op (car tmp))
   (setq arg (cadr tmp))
   (or (|npDefTail|) (|npTrap|))
   (setq body (|npPop1|))
   (if (null arg)
    (|npPush| (|pfMacro| op body))
    (|npPush| (|pfMacro| op (|pfPushMacroBody| arg body)))))))
```

————-

7.0.280 defun npMDEF

TPDHERE: Beware that this function occurs with lowercase also [npBackTrack p146]
[npStatement p168]
[npMDEFinition p163]

— **defun npMDEF** —

```
(defun |npMDEF| ()
 (|npBackTrack| #'|npStatement| 'mdef #'|npMDEFinition|))
```

————-

7.0.281 defun npMDEFinition

[npPP p207]
[npMdef p162]

— **defun npMDEFinition** —

```
(defun |npMDEFinition| ()
 (|npPP| #'|npMdef|))
```

———-

7.0.282 defun npFix

[npEqKey p143]
[npDef p185]
[npPush p141]
[pfFix p263]
[npPop1 p142]

— **defun npFix** —

```
(defun |npFix| ()
 (and
  (|npEqKey| 'fix)
  (|npPP| #'|npDef|)
  (|npPush| (|pfFix| (|npPop1|)))))
```

———-

7.0.283 defun npLet

[npLetQualified p164]
[npDefinitionOrStatement p145]

— **defun npLet** —

```
(defun |npLet| ()
 (|npLetQualified| #'|npDefinitionOrStatement|))
```

———-

7.0.284 defun npLetQualified

[npEqKey p143]
[npDefinition p165]
[npTrap p210]
[npCompMissing p149]
[npPush p141]

[pfWhere p292]
[npPop2 p142]
[npPop1 p142]

— **defun npLetQualified** —

```
(defun |npLetQualified| (f)
 (and
  (|npEqKey| 'let)
  (or (|npDefinition|) (|npTrap|))
  (|npCompMissing| 'in)
  (or (funcall f) (|npTrap|))
  (|npPush| (|pfWhere| (|npPop2|) (|npPop1|)))))
```

————-

7.0.285 defun npDefinition

[npPP p207]
[npDefinitionItem p165]
[npPush p141]
[pfSequenceToList p236]
[npPop1 p142]

— **defun npDefinition** —

```
(defun |npDefinition| ()
 (and
  (|npPP| #'|npDefinitionItem|)
  (|npPush| (|pfSequenceToList| (|npPop1|)))))
```

————-

7.0.286 defun npDefinitionItem

[npTyping p166]
[npImport p178]
[npState p210]
[npStatement p168]
[npEqPeek p150]
[npRestore p150]
[npDef p185]
[npMacro p162]
[npDefn p185]

[npTrap p210]

— **defun npDefinitionItem** —

```
(defun |npDefinitionItem| ()
 (let (a)
  (or (|npTyping|)
      (|npImport|)
      (progn
       (setq a (|npState|))
       (cond
        ((|npStatement|)
         (cond
           ((|npEqPeek| 'def)
            (|npRestore| a)
            (|npDef|))
           (t
            (|npRestore| a)
            (or (|npMacro|) (|npDefn|)))))
        (t (|npTrap|)))))))
```

————-

7.0.287 defun npTyping

[npEqKey p143]
[npDefaultItemlist p166]
[npTrap p210]
[npPush p141]
[pfTyping p289]
[npPop1 p142]

— **defun npTyping** —

```
(defun |npTyping| ()
 (and
  (|npEqKey| 'default)
  (or (|npDefaultItemlist|) (|npTrap|))
  (|npPush| (|pfTyping| (|npPop1|)))))
```

————-

7.0.288 defun npDefaultItemlist

[npPC p??]
[npSDefaultItem p167]

[npPush p141]
[pfUnSequence p291]
[npPop1 p142]

— **defun npDefaultItemlist** —

```
(defun |npDefaultItemlist| ()
 (and
  (|npPC| #'|npSDefaultItem|)
  (|npPush| (|pfUnSequence| (|npPop1|)))))
```

————-

7.0.289 defun npSDefaultItem

[npListing p153]
[npDefaultItem p167]
[npPush p141]
[pfAppend p253]
[pfParts p247]
[npPop1 p142]

— **defun npSDefaultItem** —

```
(defun |npSDefaultItem| ()
 (and
  (|npListing| #'|npDefaultItem|)
  (|npPush| (|pfAppend| (|pfParts| (|npPop1|))))))
```

————-

7.0.290 defun npDefaultItem

[npTypeVariable p154]
[npDefaultDecl p168]
[npTrap p210]

— **defun npDefaultItem** —

```
(defun |npDefaultItem| ()
 (and
  (|npTypeVariable|)
  (or (|npDefaultDecl|) (|npTrap|))))
```

————-

7.0.291 defun npDefaultDecl

[npEqKey p143]
[npType p147]
[npTrap p210]
[npPush p141]
[pfSpread p237]
[pfParts p247]
[npPop2 p142]
[npPop1 p142]

— **defun npDefaultDecl** —

```
(defun |npDefaultDecl| ()
 (and
  (|npEqKey| 'colon)
  (or (|npType|) (|npTrap|))
  (|npPush| (|pfSpread| (|pfParts| (|npPop2|)) (|npPop1|)))))
```

———-

7.0.292 defun npStatement

[npExpress p177]
[npLoop p173]
[npIterate p172]
[npReturn p176]
[npBreak p172]
[npFree p171]
[npImport p178]
[npInline p172]
[npLocal p171]
[npExport p169]
[npTyping p166]
[npVoid p177]

— **defun npStatement** —

```
(defun |npStatement| ()
 (or
  (|npExpress|)
  (|npLoop|)
  (|npIterate|)
  (|npReturn|)
  (|npBreak|)
  (|npFree|)
```

```
  (|npImport|)
  (|npInline|)
  (|npLocal|)
  (|npExport|)
  (|npTyping|)
  (|npVoid|)))
```

———-

7.0.293 defun npExport

[npEqKey p143]
[npLocalItemlist p169]
[npTrap p210]
[npPush p141]
[pfExport p262]
[npPop1 p142]

— **defun npExport** —

```
(defun |npExport| ()
 (and
  (|npEqKey| 'export)
  (or (|npLocalItemlist|) (|npTrap|))
  (|npPush| (|pfExport| (|npPop1|)))))
```

———-

7.0.294 defun npLocalItemlist

[npPC p??]
[npSLocalItem p170]
[npPush p141]
[pfUnSequence p291]
[npPop1 p142]

— **defun npLocalItemlist** —

```
(defun |npLocalItemlist| ()
 (and
  (|npPC| #'|npSLocalItem|)
  (|npPush| (|pfUnSequence| (|npPop1|)))))
```

———-

7.0.295 defun npSLocalItem

[npListing p153]
[npLocalItem p170]
[npPush p141]
[pfAppend p253]
[pfParts p247]
[npPop1 p142]

— **defun npSLocalItem** —

```
(defun |npSLocalItem| ()
 (and
  (|npListing| #'|npLocalItem|)
  (|npPush| (|pfAppend| (|pfParts| (|npPop1|))))))
```

————-

7.0.296 defun npLocalItem

[npTypeVariable p154]
[npLocalDecl p170]

— **defun npLocalItem** —

```
(defun |npLocalItem| ()
 (and
  (|npTypeVariable|)
  (|npLocalDecl|)))))
```

————-

7.0.297 defun npLocalDecl

[npEqKey p143]
[npType p147]
[npTrap p210]
[npPush p141]
[pfSpread p237]
[pfParts p247]
[npPop2 p142]
[npPop1 p142]
[pfNothing p243]

— **defun npLocalDecl** —

```
(defun |npLocalDecl| ()
 (or
  (and
   (|npEqKey| 'colon)
   (or (|npType|) (|npTrap|))
   (|npPush| (|pfSpread| (|pfParts| (|npPop2|)) (|npPop1|))))
  (|npPush| (|pfSpread| (|pfParts| (|npPop1|)) (|pfNothing|)))))
```

———-

7.0.298 defun npLocal

[npEqKey p143]
[npLocalItemlist p169]
[npTrap p210]
[npPush p141]
[pfLocal p272]
[npPop1 p142]

— **defun npLocal** —

```
(defun |npLocal| ()
 (and
  (|npEqKey| '|local|)
  (or (|npLocalItemlist|) (|npTrap|))
  (|npPush| (|pfLocal| (|npPop1|)))))
```

———-

7.0.299 defun npFree

[npEqKey p143]
[npLocalItemlist p169]
[npTrap p210]
[npPush p141]
[pfFree p263]
[npPop1 p142]

— **defun npFree** —

```
(defun |npFree| ()
```

```
(and
 (|npEqKey| 'free)
 (or (|npLocalItemlist|) (|npTrap|))
 (|npPush| (|pfFree| (|npPop1|)))))
```

———-

7.0.300 defun npInline

[npAndOr p179]
[npQualTypelist p178]
[pfInline p270]

— **defun npInline** —

```
(defun |npInline| ()
 (|npAndOr| 'inline #'|npQualTypelist| #'|pfInline|))
```

———-

7.0.301 defun npIterate

[npEqKey p143]
[npPush p141]
[pfIterate p269]
[pfNothing p243]

— **defun npIterate** —

```
(defun |npIterate| ()
 (and (|npEqKey| 'iterate) (|npPush| (|pfIterate| (|pfNothing|)))))
```

———-

7.0.302 defun npBreak

[npEqKey p143]
[npPush p141]
[pfBreak p256]
[pfNothing p243]

— **defun npBreak** —

```
(defun |npBreak| ()
 (and (|npEqKey| 'break) (|npPush| (|pfBreak| (|pfNothing|)))))
```

———-

7.0.303 defun npLoop

[npIterators p173]
[npCompMissing p149]
[npAssign p214]
[npTrap p210]
[npPush p141]
[pfLp p274]
[npPop2 p142]
[npPop1 p142]
[npEqKey p143]
[pfLoop1 p273]

— **defun npLoop** —

```
(defun |npLoop| ()
 (or
  (and
   (|npIterators|)
   (|npCompMissing| 'repeat)
   (or (|npAssign|) (|npTrap|))
   (|npPush| (|pfLp| (|npPop2|) (|npPop1|))))
  (and
   (|npEqKey| 'repeat)
   (or (|npAssign|) (|npTrap|))
   (|npPush| (|pfLoop1| (|npPop1|))))))
```

———-

7.0.304 defun npIterators

[npForIn p175]
[npZeroOrMore p175]
[npIterator p174]
[npPush p141]
[npPop2 p142]
[npPop1 p142]
[npWhile p175]
[npIterators p173]

— **defun npIterators** —

```
(defun |npIterators| ()
 (or
  (and
   (|npForIn|)
   (|npZeroOrMore| #'|npIterator|)
   (|npPush| (cons (|npPop2|) (|npPop1|))))
  (and
   (|npWhile|)
   (or
    (and (|npIterators|) (|npPush| (cons (|npPop2|) (|npPop1|))))
    (|npPush| (list (|npPop1|)))))))
```

———-

7.0.305 defun npIterator

[npForIn p175]
[npSuchThat p174]
[npWhile p175]

— **defun npIterator** —

```
(defun |npIterator| ()
 (or
  (|npForIn|)
  (|npSuchThat|)
  (|npWhile|)))
```

———-

7.0.306 defun npSuchThat

[npAndOr p179]
[npLogical p195]
[pfSuchthat p286]

— **defun npSuchThat** —

```
(defun |npSuchThat| ()
 (|npAndOr| 'bar #'|npLogical| #'|pfSuchthat|))
```

———-

7.0.307 defun Apply argument 0 or more times

[npPush p141]
[npPop2 p142]
[npPop1 p142]
[$stack p??]

— **defun npZeroOrMore** —

```
(defun |npZeroOrMore| (f)
 (let (a)
 (declare (special |$stack|))
  (cond
   ((apply f nil)
    (setq a |$stack|)
    (setq |$stack| nil)
    (do () ((not (apply f nil)))) ; while .. do
    (setq |$stack| (cons (nreverse |$stack|) a))
    (|npPush| (cons (|npPop2|) (|npPop1|))))
   (t (progn (|npPush| nil) t)))))
```

————-

7.0.308 defun npWhile

[npAndOr p179]
[npLogical p195]
[pfWhile p293]

— **defun npWhile** —

```
(defun |npWhile| ()
 (|npAndOr| 'while  #'|npLogical| #'|pfWhile|))
```

————-

7.0.309 defun npForIn

[npEqKey p143]
[npVariable p211]
[npTrap p210]
[npCompMissing p149]
[npBy p197]
[npPush p141]

[pfForin p264]
[npPop2 p142]
[npPop1 p142]

— **defun npForIn** —

```
(defun |npForIn| ()
 (and
  (|npEqKey| 'for)
  (or (|npVariable|) (|npTrap|))
  (|npCompMissing| 'in)
  (or (|npBy|) (|npTrap|))
  (|npPush| (|pfForin| (|npPop2|) (|npPop1|)))))
```

————-

7.0.310 defun npReturn

[npEqKey p143]
[npExpress p177]
[npPush p141]
[pfNothing p243]
[npEqKey p143]
[npName p202]
[npTrap p210]
[pfReturn p282]
[npPop2 p142]
[npPop1 p142]
[pfReturnNoName p283]

— **defun npReturn** —

```
(defun |npReturn| ()
 (and
  (|npEqKey| 'return)
  (or
   (|npExpress|)
   (|npPush| (|pfNothing|)))
  (or
   (and
    (|npEqKey| 'from)
    (or (|npName|) (|npTrap|))
    (|npPush| (|pfReturn| (|npPop2|) (|npPop1|))))
   (|npPush| (|pfReturnNoName| (|npPop1|))))))
```

————-

7.0.311 defun npVoid

[npAndOr p179]
[npStatement p168]
[pfNovalue p277]

— **defun npVoid** —

```
(defun |npVoid| ()
 (|npAndOr| 'do #'|npStatement| #'|pfNovalue|))
```

————-

7.0.312 defun npExpress

[npExpress1 p177]
[npIterators p173]
[npPush p141]
[pfCollect p258]
[npPop2 p142]
[pfListOf p243]
[npPop1 p142]

— **defun npExpress** —

```
(defun |npExpress| ()
 (and
  (|npExpress1|)
  (or
   (and
    (|npIterators|)
    (|npPush| (|pfCollect| (|npPop2|) (|pfListOf| (|npPop1|)))))
   t)))
```

————-

7.0.313 defun npExpress1

[npConditionalStatement p178]
[npADD p156]

— **defun npExpress1** —

```
(defun |npExpress1| ()
```

```
 (or (|npConditionalStatement|) (|npADD|)))
```

———-

7.0.314 defun npConditionalStatement

[npConditional p193]
[npQualifiedDefinition p145]

— **defun npConditionalStatement** —

```
(defun |npConditionalStatement| ()
 (|npConditional| #'|npQualifiedDefinition|))
```

———-

7.0.315 defun npImport

[npAndOr p179]
[npQualTypelist p178]
[pfImport p269]

— **defun npImport** —

```
(defun |npImport| ()
 (|npAndOr| 'import #'|npQualTypelist| #'|pfImport|))
```

———-

7.0.316 defun npQualTypelist

[npPC p**??**]
[npSQualTypelist p179]
[npPush p141]
[pfUnSequence p291]
[npPop1 p142]

— **defun npQualTypelist** —

```
(defun |npQualTypelist| ()
 (and
  (|npPC| #'|npSQualTypelist|)
```

```
  (|npPush| (|pfUnSequence| (|npPop1|)))))
```

———-

7.0.317 defun npSQualTypelist

[npListing p153]
[npQualType p179]
[npPush p141]
[pfParts p247]
[npPop1 p142]

— **defun npSQualTypelist** —

```
(defun |npSQualTypelist| ()
 (and
  (|npListing| #'|npQualType|)
  (|npPush| (|pfParts| (|npPop1|)))))
```

———-

7.0.318 defun npQualType

[npType p147]
[npPush p141]
[pfQualType p280]
[npPop1 p142]
[pfNothing p243]

— **defun npQualType** —

```
(defun |npQualType| ()
 (and
  (|npType|)
  (|npPush| (|pfQualType| (|npPop1|) (|pfNothing|)))))
```

———-

7.0.319 defun npAndOr

[npEqKey p143]
[npTrap p210]

[npPush p141]
[npPop1 p142]

— **defun npAndOr** —

```
(defun |npAndOr| (keyword p f)
 (and
  (|npEqKey| keyword)
  (or (apply p nil) (|npTrap|))
  (|npPush| (funcall f (|npPop1|)))))
```

————-

7.0.320 defun npEncAp

[npAnyNo p160]
[npEncl p180]
[npFromdom p200]

— **defun npEncAp** —

```
(defun |npEncAp| (f)
 (and (apply f nil) (|npAnyNo| #'|npEncl|) (|npFromdom|)))
```

————-

7.0.321 defun npEncl

[npBDefinition p183]
[npPush p141]
[pfApplication p251]
[npPop2 p142]
[npPop1 p142]

— **defun npEncl** —

```
(defun |npEncl| ()
 (and
  (|npBDefinition|)
  (|npPush| (|pfApplication| (|npPop2|) (|npPop1|)))))
```

————-

7.0.322 defun npAtom1

[npPDefinition p181]
[npName p202]
[npConstTok p182]
[npDollar p181]
[npBDefinition p183]
[npFromdom p200]

— **defun npAtom1** —

```
(defun |npAtom1| ()
 (or
  (|npPDefinition|)
  (and
   (or (|npName|) (|npConstTok|) (|npDollar|) (|npBDefinition|))
   (|npFromdom|))))
```

———-

7.0.323 defun npPDefinition

[npParenthesized p212]
[npDefinitionlist p191]
[npPush p141]
[pfEnSequence p261]
[npPop1 p142]

— **defun npPDefinition** —

```
(defun |npPDefinition| ()
 (and
  (|npParenthesized| #'|npDefinitionlist|)
  (|npPush| (|pfEnSequence| (|npPop1|)))))
```

———-

7.0.324 defun npDollar

[npEqPeek p150]
[npPush p141]
[tokConstruct p409]
[tokPosn p411]
[npNext p143]

[$stok p**??**]

— **defun npDollar** —

```
(defun |npDollar| ()
 (declare (special |$stok|))
  (and (|npEqPeek| '$)
       (progn
        (|npPush| (|tokConstruct| '|id| '$ (|tokPosn| |$stok|)))
        (|npNext|))))
```

———-

7.0.325 defun npConstTok

[tokType p411]
[npPush p141]
[npNext p143]
[npEqPeek p150]
[npState p210]
[npPrimary1 p162]
[pfSymb p249]
[npPop1 p142]
[tokPosn p411]
[npRestore p150]
[$stok p**??**]

— **defun npConstTok** —

```
(defun |npConstTok| ()
 (let (b a)
 (declare (special |$stok|))
  (cond
   ((member (|tokType| |$stok|) '(|integer| |string| |char| |float| |command|))
    (|npPush| |$stok|)
    (|npNext|))
   ((|npEqPeek| '|'|)
    (setq a |$stok|)
    (setq b (|npState|))
    (|npNext|)
    (cond
      ((and (|npPrimary1|)
            (|npPush| (|pfSymb| (|npPop1|) (|tokPosn| a))))
       t)
      (t (|npRestore| b) nil)))
   (t nil))))
```

———-

7.0.326 defun npBDefinition

[npPDefinition p181]
[npBracketed p183]
[npDefinitionlist p191]

— **defun npBDefinition** —

```
(defun |npBDefinition| ()
 (or
  (|npPDefinition|)
  (|npBracketed| #'|npDefinitionlist|)))
```

———-

7.0.327 defun npBracketed

[npParened p183]
[npBracked p184]
[npBraced p184]
[npAngleBared p184]

— **defun npBracketed** —

```
(defun |npBracketed| (f)
 (or
  (|npParened| f)
  (|npBracked| f)
  (|npBraced| f)
  (|npAngleBared| f)))
```

———-

7.0.328 defun npParened

[npEnclosed p209]
[pfParen p279]

— **defun npParened** —

```
(defun |npParened| (f)
```

```
 (or (|npEnclosed| '|(| '|)| #'|pfParen| f)
     (|npEnclosed| '|(\|| '|\|)| #'|pfParen| f)))
```

———-

7.0.329 defun npBracked

[npEnclosed p209]
[pfBracket p255]
[pfBracketBar p255]

— **defun npBracked** —

```
(defun |npBracked| (f)
 (or (|npEnclosed| '[ '] #'|pfBracket| f)
     (|npEnclosed| '|[\|| '|\|]| #'|pfBracketBar| f)))
```

———-

7.0.330 defun npBraced

[npEnclosed p209]
[pfBrace p255]
[pfBraceBar p255]

— **defun npBraced** —

```
(defun |npBraced| (f)
 (or (|npEnclosed| '{ '} #'|pfBrace| f)
     (|npEnclosed| '|{\|| '|\|}| #'|pfBraceBar| f)))
```

———-

7.0.331 defun npAngleBared

[npEnclosed p209]
[pfHide p267]

— **defun npAngleBared** —

```
(defun |npAngleBared| (f)
  (|npEnclosed| '|<\|| '|\|>| #'|pfHide| f))
```

———-

7.0.332 defun npDefn

[npEqKey p143]
[npPP p207]
[npDef p185]

— **defun npDefn** —

```
(defun |npDefn| ()
 (and
  (|npEqKey| 'defn)
  (|npPP| #'|npDef|)))
```

———-

7.0.333 defun npDef

[npMatch p148]
[pfCheckItOut p237]
[npPop1 p142]
[npDefTail p192]
[npTrap p210]
[npPop1 p142]
[npPush p141]
[pfDefinition p259]
[pfPushBody p247]

— **defun npDef** —

```
(defun |npDef| ()
 (let (body rt arg op tmp1)
  (when (|npMatch|)
   ; [op,arg,rt]:=  pfCheckItOut(npPop1())
   (setq tmp1 (|pfCheckItOut| (|npPop1|)))
   (setq op (car tmp1))
   (setq arg (cadr tmp1))
   (setq rt (caddr tmp1))
   (or (|npDefTail|) (|npTrap|))
   (setq body (|npPop1|))
   (if (null arg)
    (|npPush| (|pfDefinition| op body))
    (|npPush| (|pfDefinition| op (|pfPushBody| rt arg body)))))))
```

———-

7.0.334 defun npBPileDefinition

[npPileBracketed p186]
[npPileDefinitionlist p187]
[npPush p141]
[pfSequence p285]
[pfListOf p243]
[npPop1 p142]

— **defun npBPileDefinition** —

```
(defun |npBPileDefinition| ()
 (and
  (|npPileBracketed| #'|npPileDefinitionlist|)
  (|npPush| (|pfSequence| (|pfListOf| (|npPop1|))))))
```

———-

7.0.335 defun npPileBracketed

[npEqKey p143]
[npPush p141]
[pfNothing p243]
[npMissing p149]
[pfPile p247]
[npPop1 p142]

— **defun npPileBracketed** —

```
(defun |npPileBracketed| (f)
 (cond
  ((|npEqKey| 'settab)
   (cond
    ((|npEqKey| 'backtab) (|npPush| (|pfNothing|))) ; never happens
    ((and (apply f nil)
          (or (|npEqKey| 'backtab) (|npMissing| '|backtab|)))
     (|npPush| (|pfPile| (|npPop1|))))
    (t nil)))
  (t nil)))
```

———-

7.0.336 defun npPileDefinitionlist

[npListAndRecover p187]
[npDefinitionlist p191]
[npPush p141]
[pfAppend p253]
[npPop1 p142]

— **defun npPileDefinitionlist** —

```
(defun |npPileDefinitionlist| ()
 (and
  (|npListAndRecover| #'|npDefinitionlist|)
  (|npPush| (|pfAppend| (|npPop1|)))))
```

————-

7.0.337 defun npListAndRecover

[trappoint p**??**]
[npRecoverTrap p188]
[syGeneralErrorHere p190]
[npEqKey p143]
[npEqPeek p150]
[npNext p143]
[npPop1 p142]
[npPush p141]
[$inputStream p**??**]
[$stack p**??**]

— **defun npListAndRecover** —

```
(defun |npListAndRecover| (f)
 (let (found c done b savestack)
 (declare (special |$inputStream| |$stack|))
  (setq savestack |$stack|)
  (setq |$stack| nil)
  (setq c |$inputStream|)
  (do ()
      (done)
   (setq found (catch 'trappoint (apply f nil)))
   (cond
    ((eq found 'trapped)
     (setq |$inputStream| c)
     (|npRecoverTrap|))
    ((null found)
```

```
    (setq |$inputStream| c)
    (|syGeneralErrorHere|) (|npRecoverTrap|)))
  (cond
   ((|npEqKey| 'backset) (setq c |$inputStream|))
   ((|npEqPeek| 'backtab) (setq done t))
   (t
    (setq |$inputStream| c)
    (|syGeneralErrorHere|)
    (|npRecoverTrap|)
    (cond
     ((|npEqPeek| 'backtab) (setq done t))
     (t
      (|npNext|)
      (setq c |$inputStream|)))))
  (setq b (cons (|npPop1|) b)))
 (setq |$stack| savestack)
 (|npPush| (nreverse b))))
```

———-

7.0.338 defun npRecoverTrap

[npFirstTok p141]
[tokPosn p411]
[npMoveTo p189]
[syIgnoredFromTo p189]
[npPush p141]
[pfWrong p294]
[pfDocument p244]
[pfListOf p243]
[$stok p??]

— **defun npRecoverTrap** —

```
(defun |npRecoverTrap| ()
 (let (pos2 pos1)
 (declare (special |$stok|))
  (|npFirstTok|)
  (setq pos1 (|tokPosn| |$stok|))
  (|npMoveTo| 0)
  (setq pos2 (|tokPosn| |$stok|))
  (|syIgnoredFromTo| pos1 pos2)
  (|npPush|
   (list (|pfWrong| (|pfDocument| (list "pile syntax error"))
                    (|pfListOf| nil))))))
```

———-

7.0.339 defun npMoveTo

[npEqPeek p150]
[npNext p143]
[npMoveTo p189]
[npEqKey p143]
[$inputStream p??]

— **defun npMoveTo** —

```
(defun |npMoveTo| (|n|)
 (declare (special |$inputStream|))
  (cond
   ((null |$inputStream|) t)
   ((|npEqPeek| 'backtab)
    (cond
     ((eql |n| 0) t)
     (t (|npNext|) (|npMoveTo| (1- |n|)))))
   ((|npEqPeek| 'backset)
    (cond
     ((eql |n| 0) t)
     (t (|npNext|) (|npMoveTo| |n|))))
   ((|npEqKey| 'settab) (|npMoveTo| (+ |n| 1)))
   (t (|npNext|) (|npMoveTo| |n|))))
```

———-

7.0.340 defun syIgnoredFromTo

[pfGlobalLinePosn p233]
[ncSoftError p349]
[FromTo p378]
[From p378]
[To p378]

— **defun syIgnoredFromTo** —

```
(defun |syIgnoredFromTo| (pos1 pos2)
 (cond
  ((equal (|pfGlobalLinePosn| pos1) (|pfGlobalLinePosn| pos2))
   (|ncSoftError| (|FromTo| pos1 pos2) 'S2CY0005 nil))
  (t
   (|ncSoftError| (|From| pos1) 'S2CY0003 nil)
   (|ncSoftError| (|To| pos2) 'S2CY0004 nil))))
```

———-

7.0.341 defun syGeneralErrorHere

[sySpecificErrorHere p190]

— **defun syGeneralErrorHere** —

```
(defun |syGeneralErrorHere| ()
 (|sySpecificErrorHere| 'S2CY0002 nil))
```

———-

7.0.342 defun sySpecificErrorHere

[sySpecificErrorAtToken p190]
[$stok p**??**]

— **defun sySpecificErrorHere** —

```
(defun |sySpecificErrorHere| (key args)
 (declare (special |$stok|))
 (|sySpecificErrorAtToken| |$stok| key args))
```

———-

7.0.343 defun sySpecificErrorAtToken

[ncSoftError p349]
[tokPosn p411]

— **defun sySpecificErrorAtToken** —

```
(defun |sySpecificErrorAtToken| (tok key args)
 (|ncSoftError| (|tokPosn| tok) key args))
```

———-

7.0.344 defun npDefinitionlist

[npSemiListing p191]
[npQualDef p143]

— **defun npDefinitionlist** —

```
(defun |npDefinitionlist| ()
 (|npSemiListing| #'|npQualDef|))
```

————-

7.0.345 defun npSemiListing

[npListofFun p219]
[npSemiBackSet p191]
[pfAppend p253]

— **defun npSemiListing** —

```
(defun |npSemiListing| (p)
 (|npListofFun| p #'|npSemiBackSet| #'|pfAppend|))
```

————-

7.0.346 defun npSemiBackSet

[npEqKey p143]

— **defun npSemiBackSet** —

```
(defun |npSemiBackSet| ()
 (and (|npEqKey| 'semicolon) (or (|npEqKey| 'backset) t)))
```

————-

7.0.347 defun npRule

[npEqKey p143]
[npPP p207]
[npSingleRule p192]

— **defun npRule** —

```
(defun |npRule| ()
 (and
  (|npEqKey| 'rule)
  (|npPP| #'|npSingleRule|)))
```

———-

7.0.348 defun npSingleRule

[npQuiver p196]
[npDefTail p192]
[npTrap p210]
[npPush p141]
[pfRule p283]
[npPop2 p142]
[npPop1 p142]

— **defun npSingleRule** —

```
(defun |npSingleRule| ()
 (when (|npQuiver|)
  (or (|npDefTail|) (|npTrap|))
  (|npPush| (|pfRule| (|npPop2|) (|npPop1|)))))
```

———-

7.0.349 defun npDefTail

[npEqKey p143]
[npDefinitionOrStatement p145]

— **defun npDefTail** —

```
(defun |npDefTail| ()
 (and
  (or (|npEqKey| 'def) (|npEqKey| 'mdef))
  (|npDefinitionOrStatement|)))
```

———-

7.0.350 defun npDefaultValue

[npEqKey p143]
[npDefinitionOrStatement p145]

[npTrap p210]
[npPush p141]
[pfAdd p250]
[pfNothing p243]
[npPop1 p142]

— **defun npDefaultValue** —

```
(defun |npDefaultValue| ()
 (and
  (|npEqKey| 'default)
  (or (|npDefinitionOrStatement|) (|npTrap|))
  (|npPush| (list (|pfAdd| (|pfNothing|) (|npPop1|) (|pfNothing|))))))
```

———-

7.0.351 defun npWConditional

[npConditional p193]
[npPush p141]
[pfTweakIf p288]
[npPop1 p142]

— **defun npWConditional** —

```
(defun |npWConditional| (f)
 (when (|npConditional| f) (|npPush| (|pfTweakIf| (|npPop1|)))))
```

———-

7.0.352 defun npConditional

[npEqKey p143]
[npLogical p195]
[npTrap p210]
[npMissing p149]
[npElse p194]

— **defun npConditional** —

```
(defun |npConditional| (f)
 (cond
  ((and (|npEqKey| 'IF)
        (or (|npLogical|) (|npTrap|))
```

```
       (or (|npEqKey| 'backset) t))
  (cond
   ((|npEqKey| 'settab)
    (cond
     ((|npEqKey| 'then)
      (and (or (apply f nil) (|npTrap|))
           (|npElse| f)
           (|npEqKey| 'backtab)))
     (t (|npMissing| '|then|))))
   ((|npEqKey| 'then)
    (and (or (apply f nil) (|npTrap|)) (|npElse| f)))
   (t (|npMissing| '|then|))))
 (t nil)))
```

———-

7.0.353 defun npElse

[npState p210]
[npBacksetElse p195]
[npTrap p210]
[npPush p141]
[pfIf p267]
[npPop3 p142]
[npPop2 p142]
[npPop1 p142]
[npRestore p150]
[pfIfThenOnly p268]

— **defun npElse** —

```
(defun |npElse| (f)
 (let (a)
  (setq a (|npState|))
  (cond
   ((|npBacksetElse|)
    (and
     (or (apply f nil) (|npTrap|))
     (|npPush| (|pfIf| (|npPop3|) (|npPop2|) (|npPop1|)))))
   (t
     (|npRestore| a)
     (|npPush| (|pfIfThenOnly| (|npPop2|) (|npPop1|)))))))
```

———-

7.0.354 defun npBacksetElse

TPDHERE: Well this makes no sense. [npEqKey p143]

— **defun npBacksetElse** —

```
(defun |npBacksetElse| ()
 (if (|npEqKey| 'backset)
  (|npEqKey| 'else)
  (|npEqKey| 'else)))
```

————-

7.0.355 defun npLogical

[npLeftAssoc p204]
[npDisjand p195]

— **defun npLogical** —

```
(defun |npLogical| ()
 (|npLeftAssoc| '(or) #'|npDisjand|))
```

————-

7.0.356 defun npDisjand

[npLeftAssoc p204]
[npDiscrim p195]

— **defun npDisjand** —

```
(defun |npDisjand| ()
 (|npLeftAssoc| '(and) #'|npDiscrim|))
```

————-

7.0.357 defun npDiscrim

[npLeftAssoc p204]
[npQuiver p196]

— **defun npDiscrim** —

```
(defun |npDiscrim| ()
 (|npLeftAssoc| '(case has) #'|npQuiver|))
```

———-

7.0.358 defun npQuiver

[npRightAssoc p204]
[npRelation p196]

— **defun npQuiver** —

```
(defun |npQuiver| ()
 (|npRightAssoc| '(arrow larrow) #'|npRelation|))
```

———-

7.0.359 defun npRelation

[npLeftAssoc p204]
[npSynthetic p196]

— **defun npRelation** —

```
(defun |npRelation| ()
 (|npLeftAssoc| '(equal notequal lt le gt ge oangle cangle) #'|npSynthetic|))
```

———-

7.0.360 defun npSynthetic

[npBy p197]
[npAmpersandFrom p200]
[npPush p141]
[pfApplication p251]
[npPop2 p142]
[npPop1 p142]
[pfInfApplication p269]

— **defun npSynthetic** —

```
(defun |npSynthetic| ()
```

```
(cond
 ((|npBy|)
  ((lambda ()
   (loop
    (cond
     ((not (and (|npAmpersandFrom|)
                (or (|npBy|)
                    (progn
                     (|npPush| (|pfApplication| (|npPop2|) (|npPop1|)))
                     nil))))
        (return nil))
     (t
      (|npPush| (|pfInfApplication| (|npPop2|) (|npPop2|) (|npPop1|)))))))
      t)
  (t nil)))
```

———-

7.0.361 defun npBy

[npLeftAssoc p204]
[npInterval p197]

— **defun npBy** —

```
(defun |npBy| ()
 (|npLeftAssoc| '(by) #'|npInterval|))
```

———-

7.0.362 defun

[npArith p198]
[npSegment p198]
[npEqPeek p150]
[npPush p141]
[pfApplication p251]
[npPop1 p142]
[pfInfApplication p269]
[npPop2 p142]

— **defun npInterval** —

```
(defun |npInterval| ()
```

```
(and
 (|npArith|)
 (or
  (and
   (|npSegment|)
   (or
    (and
     (|npEqPeek| 'bar)
     (|npPush| (|pfApplication| (|npPop1|) (|npPop1|))))
    (and
     (|npArith|)
     (|npPush| (|pfInfApplication| (|npPop2|) (|npPop2|) (|npPop1|))))
    (|npPush| (|pfApplication| (|npPop1|) (|npPop1|)))))
  t)))
```

———-

7.0.363 defun npSegment

[npEqPeek p150]
[npPushId p207]
[npFromdom p200]

— **defun npSegment** —

```
(defun |npSegment| ()
 (and (|npEqPeek| 'seg) (|npPushId|) (|npFromdom|)))
```

———-

7.0.364 defun npArith

[npLeftAssoc p204]
[npSum p199]

— **defun npArith** —

```
(defun |npArith| ()
 (|npLeftAssoc| '(mod) #'|npSum|))
```

———-

7.0.365 defun npSum

[npLeftAssoc p204]
[npTerm p199]

— **defun npSum** —

```
(defun |npSum| ()
 (|npLeftAssoc| '(plus minus) #'|npTerm|))
```

————-

7.0.366 defun npTerm

[npInfGeneric p205]
[npRemainder p199]
[npPush p141]
[pfApplication p251]
[npPop2 p142]
[npPop1 p142]

— **defun npTerm** —

```
(defun |npTerm| ()
 (or
  (and
   (|npInfGeneric| '(minus plus))
   (or
    (and (|npRemainder|) (|npPush| (|pfApplication| (|npPop2|) (|npPop1|))))
    t))
  (|npRemainder|)))
```

————-

7.0.367 defun npRemainder

[npLeftAssoc p204]
[npProduct p200]

— **defun npRemainder** —

```
(defun |npRemainder| ()
 (|npLeftAssoc| '(rem quo) #'|npProduct|))
```

————-

7.0.368 defun npProduct

[npLeftAssoc p204]
[npPower p200]

— **defun npProduct** —

```
(defun |npProduct| ()
 (|npLeftAssoc|
  '(times slash backslash slashslash backslashbackslash
          slashbackslash backslashslash)
    #'|npPower|))
```

———-

7.0.369 defun npPower

[npRightAssoc p204]
[npColon p215]

— **defun npPower** —

```
(defun |npPower| ()
 (|npRightAssoc| '(power carat) #'|npColon|))
```

———-

7.0.370 defun npAmpersandFrom

[npAmpersand p202]
[npFromdom p200]

— **defun npAmpersandFrom** —

```
(defun |npAmpersandFrom| ()
 (and (|npAmpersand|) (|npFromdom|)))
```

———-

7.0.371 defun npFromdom

[npEqKey p143]
[npApplication p160]

[npTrap p210]
[npFromdom1 p201]
[npPop1 p142]
[npPush p141]
[pfFromDom p265]

— **defun npFromdom** —

```
(defun |npFromdom| ()
 (or
  (and
   (|npEqKey| '$)
   (or (|npApplication|) (|npTrap|))
   (|npFromdom1| (|npPop1|))
   (|npPush| (|pfFromDom| (|npPop1|) (|npPop1|))))
  t))
```

———-

7.0.372 defun npFromdom1

[npEqKey p143]
[npApplication p160]
[npTrap p210]
[npFromdom1 p201]
[npPop1 p142]
[npPush p141]
[pfFromDom p265]

— **defun npFromdom1** —

```
(defun |npFromdom1| (c)
 (or
  (and
   (|npEqKey| '$)
   (or (|npApplication|) (|npTrap|))
   (|npFromdom1| (|npPop1|))
   (|npPush| (|pfFromDom| (|npPop1|) c)))
  (|npPush| c)))))
```

———-

7.0.373 defun npAmpersand

[npEqKey p143]
[npName p202]
[npTrap p210]

— **defun npAmpersand** —

```
(defun |npAmpersand| ()
 (and
  (|npEqKey| 'ampersand)
  (or (|npName|) (|npTrap|))))
```

———-

7.0.374 defun npName

[npId p202]
[npSymbolVariable p203]

— **defun npName** —

```
(defun |npName| ()
 (or (|npId|) (|npSymbolVariable|)))
```

———-

7.0.375 defvar $npPParg

— **initvars** —

```
(defvar |$npTokToNames| (list '~ '|#| '[] '{} '|[\|\|]| '|{\|\|}|))
```

———-

7.0.376 defun npId

[npPush p141]
[npNext p143]
[tokConstruct p409]
[tokPosn p411]

[$npTokToNames p**??**]
[$ttok p**??**]
[$stok p**??**]

— **defun npId** —

```
(defun |npId| ()
 (declare (special |$npTokToNames| |$ttok| |$stok|))
  (cond
   ((eq (caar |$stok|) '|id|)
     (|npPush| |$stok|)
     (|npNext|))
   ((and (eq (caar |$stok|) '|key|) (member |$ttok| |$npTokToNames|))
     (|npPush| (|tokConstruct| '|id| |$ttok| (|tokPosn| |$stok|)))
     (|npNext|))
   (t nil)))
```

————-

7.0.377 defun npSymbolVariable

[npState p210]
[npEqKey p143]
[npId p202]
[npPop1 p142]
[npPush p141]
[tokConstruct p409]
[tokPart p411]
[tokPosn p411]
[npRestore p150]

— **defun npSymbolVariable** —

```
(defun |npSymbolVariable| ()
 (let (a)
  (setq a (|npState|))
  (cond
   ((and (|npEqKey| 'backquote) (|npId|))
    (setq a (|npPop1|))
    (|npPush| (|tokConstruct| '|idsy| (|tokPart| a) (|tokPosn| a))))
   (t (|npRestore| a) nil))))
```

————-

7.0.378 defun npRightAssoc

[npState p210]
[npInfGeneric p205]
[npRightAssoc p204]
[npPush p141]
[pfApplication p251]
[npPop2 p142]
[npPop1 p142]
[pfInfApplication p269]
[npRestore p150]

— **defun npRightAssoc** —

```
(defun |npRightAssoc| (o p)
 (let (a)
  (setq a (|npState|))
  (cond
   ((apply p nil)
      ((lambda ()
       (loop
        (cond
         ((not
           (and
            (|npInfGeneric| o)
            (or
             (|npRightAssoc| o p)
             (progn (|npPush| (|pfApplication| (|npPop2|) (|npPop1|))) nil))))
          (return nil))
         (t
          (|npPush| (|pfInfApplication| (|npPop2|) (|npPop2|) (|npPop1|))))))
       t)
   (t
    (|npRestore| a)
    nil))))
```

———-

7.0.379 defun p o p o p o p = (((p o p) o p) o p)

```
p o p o p o p = (((p o p) o p) o p)
p o p o = (p o p) o
;npLeftAssoc(operations,parser)==
;    if APPLY(parser,nil)
;    then
;       while npInfGeneric(operations)
;         and (APPLY(parser,nil) or
```

```
;              (npPush pfApplication(npPop2(),npPop1());false))
;           repeat
;             npPush pfInfApplication(npPop2(),npPop2(),npPop1())
;       true
;    else false
```

[npInfGeneric p205]
[npPush p141]
[pfApplication p251]
[npPop2 p142]
[npPop1 p142]
[pfInfApplication p269]

— **defun npLeftAssoc** —

```
(defun |npLeftAssoc| (operations parser)
 (when (apply parser nil)
   ((lambda nil
      (loop
       (cond
        ((not
          (and
           (|npInfGeneric| operations)
           (or
            (apply parser nil)
            (progn (|npPush| (|pfApplication| (|npPop2|) (|npPop1|))) nil))))
         (return nil))
        (t
         (|npPush| (|pfInfApplication| (|npPop2|) (|npPop2|) (|npPop1|))))))))
     t))
```

———-

7.0.380 defun npInfGeneric

[npDDInfKey p206]
[npEqKey p143]

— **defun npInfGeneric** —

```
(defun |npInfGeneric| (s)
 (and
  (|npDDInfKey| s)
  (or (|npEqKey| 'backset) t)))
```

———-

7.0.381 defun npDDInfKey

[npInfKey p206]
[npState p210]
[npEqKey p143]
[npPush p141]
[pfSymb p249]
[npPop1 p142]
[tokPosn p411]
[npRestore p150]
[tokConstruct p409]
[tokPart p411]
[$stok p??]

— **defun npDDInfKey** —

```
(defun |npDDInfKey| (s)
 (let (b a)
 (declare (special |$stok|))
  (or
   (|npInfKey| s)
   (progn
    (setq a (|npState|))
    (setq b |$stok|)
    (cond
     ((and (|npEqKey| '|'|) (|npInfKey| s))
       (|npPush| (|pfSymb| (|npPop1|) (|tokPosn| b))))
     (t
      (|npRestore| a)
      (cond
       ((and (|npEqKey| 'backquote) (|npInfKey| s))
         (setq a (|npPop1|))
          (|npPush| (|tokConstruct| '|idsy| (|tokPart| a) (|tokPosn| a))))
       (t
        (|npRestore| a)
        nil))))))))
```

———-

7.0.382 defun npInfKey

[npPushId p207]
[$stok p??]
[$ttok p??]

— **defun npInfKey** —

```
(defun |npInfKey| (s)
 (declare (special |$ttok| |$stok|))
  (and (eq (caar |$stok|) '|key|) (member |$ttok| s) (|npPushId|)))
```

———-

7.0.383 defun npPushId

[tokConstruct p409]
[tokPosn p411]
[npNext p143]
[$stack p**??**]
[$stok p**??**]
[$ttok p**??**]

— **defun npPushId** —

```
(defun |npPushId| ()
 (let (a)
 (declare (special |$stack| |$stok| |$ttok|))
  (setq a (get |$ttok| 'infgeneric))
  (when a (setq |$ttok| a))
  (setq |$stack|
   (cons (|tokConstruct| '|id| |$ttok| (|tokPosn| |$stok|)) |$stack|))
  (|npNext|)))
```

———-

7.0.384 defvar $npPParg

— **initvars** —

```
(defvar *npPParg* nil "rewrite npPP without flets, using global scoping")
```

———-

7.0.385 defun npPP

This was rewritten by NAG to remove flet. [npParened p183]
[npPPf p209]
[npPileBracketed p186]

[npPPg p208]
[npPush p141]
[pfEnSequence p261]
[npPop1 p142]
[npPParg p207]

— **defun npPP** —

```
(defun |npPP| (f)
 (declare (special *npPParg*))
  (setq *npPParg* f)
  (or
   (|npParened| #'npPPf)
   (and  (|npPileBracketed| #'npPPg) (|npPush| (|pfEnSequence| (|npPop1|))))
   (funcall f)))
```

————-

7.0.386 defun npPPff

[npPop1 p142]
[npPush p141]
[$npPParg p207]

— **defun npPPff** —

```
(defun npPPff ()
 (and (funcall *npPParg*) (|npPush| (list (|npPop1|)))))
```

————-

7.0.387 defun npPPg

[npListAndRecover p187]
[npPPf p209]
[npPush p141]
[pfAppend p253]
[npPop1 p142]

— **defun npPPg** —

```
(defun npPPg ()
 (and (|npListAndRecover| #'npPPf))
 (|npPush| (|pfAppend| (|npPop1|))))
```

———-

7.0.388 defun npPPf

[npSemiListing p191]
[npPPff p208]

— **defun npPPf** —

```
(defun npPPf ()
 (|npSemiListing| #'npPPff))
```

———-

7.0.389 defun npEnclosed

[npEqKey p143]
[npPush p141]
[pfTuple p290]
[pfListOf p243]
[npMissingMate p213]
[pfEnSequence p261]
[npPop1 p142]
[$stok p**??**]

— **defun npEnclosed** —

```
(defun |npEnclosed| (open close fn f)
 (let (a)
 (declare (special |$stok|))
  (setq a |$stok|)
  (when (|npEqKey| open)
   (cond
    ((|npEqKey| close)
     (|npPush| (funcall fn a (|pfTuple| (|pfListOf| NIL)))))
    ((and (apply f nil)
          (or (|npEqKey| close)
              (|npMissingMate| close a)))
     (|npPush| (funcall fn a (|pfEnSequence| (|npPop1|)))))
    ('t nil)))))
```

———-

7.0.390 defun npState

[$stack p??]
[$inputStream p??]

— **defun npState** —

```
(defun |npState| ()
 (declare (special |$stack| |$inputStream|))
  (cons |$inputStream| |$stack|))
```

———-

7.0.391 defun npTrap

[trappoint p??]
[tokPosn p411]
[ncSoftError p349]
[$stok p??]

— **defun npTrap** —

```
(defun |npTrap| ()
 (declare (special |$stok|))
  (|ncSoftError| (|tokPosn| |$stok|) 'S2CY0002 nil)
  (throw 'trappoint 'trapped))
```

———-

7.0.392 defun npTrapForm

[trappoint p??]
[pfSourceStok p241]
[syGeneralErrorHere p190]
[ncSoftError p349]
[tokPosn p411]

— **defun npTrapForm** —

```
(defun |npTrapForm| (x)
 (let (a)
  (setq a (|pfSourceStok| x))
  (cond
   ((eq a '|NoToken|)
```

```
    (|syGeneralErrorHere|)
    (throw 'trappoint 'trapped))
   (t
    (|ncSoftError| (|tokPosn| a) 'S2CY0002 nil)
    (throw 'trappoint 'trapped)))))
```

———-

7.0.393 defun npVariable

[npParenthesized p212]
[npVariablelist p211]
[npVariableName p211]
[npPush p141]
[pfListOf p243]
[npPop1 p142]

— **defun npVariable** —

```
(defun |npVariable| ()
 (or
  (|npParenthesized| #'|npVariablelist|)
  (and (|npVariableName|) (|npPush| (|pfListOf| (list (|npPop1|)))))))
```

———-

7.0.394 defun npVariablelist

[npListing p153]
[npVariableName p211]

— **defun npVariablelist** —

```
(defun |npVariablelist| ()
 (|npListing| #'|npVariableName|))
```

———-

7.0.395 defun npVariableName

[npName p202]
[npDecl p212]

[npPush p141]
[pfTyped p288]
[npPop1 p142]
[pfNothing p243]

— **defun npVariableName** —

```
(defun |npVariableName| ()
 (and
  (|npName|)
  (or (|npDecl|) (|npPush| (|pfTyped| (|npPop1|) (|pfNothing|))))))
```

————-

7.0.396 defun npDecl

[npEqKey p143]
[npType p147]
[npTrap p210]
[npPush p141]
[pfTyped p288]
[npPop2 p142]
[npPop1 p142]

— **defun npDecl** —

```
(defun |npDecl| ()
 (and
  (|npEqKey| 'colon)
  (or (|npType|) (|npTrap|))
  (|npPush| (|pfTyped| (|npPop2|) (|npPop1|)))))
```

————-

7.0.397 defun npParenthesized

[npParenthesize p213]

— **defun npParenthesized** —

```
(defun |npParenthesized| (f)
 (or (|npParenthesize| '|(| '|)| f) (|npParenthesize| '|(\|| '|\|)| f)))
```

————-

7.0.398 defun npParenthesize

[npEqKey p143]
[npMissingMate p213]
[npPush p141]
[$stok p**??**]

— **defun npParenthesize** —

```
(defun |npParenthesize| (open close f)
 (let (a)
 (declare (special |$stok|))
  (setq a |$stok|)
  (cond
   ((|npEqKey| open)
    (cond
     ((and (apply f nil)
           (or (|npEqKey| close)
               (|npMissingMate| close a)))
      t)
     ((|npEqKey| close) (|npPush| nil))
     (t (|npMissingMate| close a))))
   (t nil))))
```

———-

7.0.399 defun npMissingMate

[ncSoftError p349]
[tokPosn p411]
[npMissing p149]

— **defun npMissingMate** —

```
(defun |npMissingMate| (close open)
 (|ncSoftError| (|tokPosn| open) 'S2CY0008 nil)
 (|npMissing| close))
```

———-

7.0.400 defun npExit

[npBackTrack p146]
[npAssign p214]

[npPileExit p214]

— **defun npExit** —

```
(defun |npExit| ()
 (|npBackTrack| #'|npAssign| 'exit #'|npPileExit|))
```

———-

7.0.401 defun npPileExit

[npAssign p214]
[npEqKey p143]
[npStatement p168]
[npPush p141]
[pfExit p261]
[npPop2 p142]
[npPop1 p142]

— **defun npPileExit** —

```
(defun |npPileExit| ()
 (and
  (|npAssign|)
  (or (|npEqKey| 'exit) (|npTrap|))
  (or (|npStatement|) (|npTrap|))
  (|npPush| (|pfExit| (|npPop2|) (|npPop1|)))))
```

———-

7.0.402 defun npAssign

[npBackTrack p146]
[npMDEF p163]
[npAssignment p215]

— **defun npAssign** —

```
(defun |npAssign| ()
 (|npBackTrack| #'|npMDEF| 'becomes #'|npAssignment|))
```

———-

7.0.403 defun npAssignment

[npAssignVariable p215]
[npEqKey p143]
[npTrap p210]
[npGives p146]
[npPush p141]
[pfAssign p253]
[npPop2 p142]
[npPop1 p142]

— **defun npAssignment** —

```
(defun |npAssignment| ()
 (and
  (|npAssignVariable|)
  (or (|npEqKey| 'becomes) (|npTrap|))
  (or (|npGives|) (|npTrap|))
  (|npPush| (|pfAssign| (|npPop2|) (|npPop1|)))))
```

———-

7.0.404 defun npAssignVariable

[npColon p215]
[npPush p141]
[pfListOf p243]
[npPop1 p142]

— **defun npAssignVariable** —

```
(defun |npAssignVariable| ()
 (and (|npColon|) (|npPush| (|pfListOf| (list (|npPop1|))))))
```

———-

7.0.405 defun npColon

[npTypified p216]
[npAnyNo p160]
[npTagged p216]

— **defun npColon** —

```
(defun |npColon| ()
 (and (|npTypified|) (|npAnyNo| #'|npTagged|)))
```

———-

7.0.406 defun npTagged

[npTypedForm1 p216]
[pfTagged p286]

— **defun npTagged** —

```
(defun |npTagged| ()
 (|npTypedForm1| 'colon #'|pfTagged|))
```

———-

7.0.407 defun npTypedForm1

[npEqKey p143]
[npType p147]
[npTrap p210]
[npPush p141]
[npPop2 p142]
[npPop1 p142]

— **defun npTypedForm1** —

```
(defun |npTypedForm1| (sy fn)
 (and
  (|npEqKey| sy)
  (or (|npType|) (|npTrap|))
  (|npPush| (funcall fn (|npPop2|) (|npPop1|)))))
```

———-

7.0.408 defun npTypified

[npApplication p160]
[npAnyNo p160]
[npTypeStyle p217]

— **defun npTypified** —

```
(defun |npTypified| ()
 (and (|npApplication|) (|npAnyNo| #'|npTypeStyle|)))
```

———-

7.0.409 defun npTypeStyle

[npCoerceTo p218]
[npRestrict p218]
[npPretend p217]
[npColonQuery p217]

— **defun npTypeStyle** —

```
(defun |npTypeStyle| ()
 (or (|npCoerceTo|) (|npRestrict|) (|npPretend|) (|npColonQuery|)))
```

———-

7.0.410 defun npPretend

[npTypedForm p218]
[pfPretend p279]

— **defun npPretend** —

```
(defun |npPretend| ()
 (|npTypedForm| 'pretend #'|pfPretend|))
```

———-

7.0.411 defun npColonQuery

[npTypedForm p218]
[pfRetractTo p282]

— **defun npColonQuery** —

```
(defun |npColonQuery| ()
 (|npTypedForm| 'atat #'|pfRetractTo|))
```

———-

7.0.412 defun npCoerceTo

[npTypedForm p218]
[pfCoerceto p257]

— **defun npCoerceTo** —

```
(defun |npCoerceTo| ()
 (|npTypedForm| 'coerce #'|pfCoerceto|))
```

———-

7.0.413 defun npTypedForm

[npEqKey p143]
[npApplication p160]
[npTrap p210]
[npPush p141]
[npPop2 p142]
[npPop1 p142]

— **defun npTypedForm** —

```
(defun |npTypedForm| (sy fn)
 (and
  (|npEqKey| sy)
  (or (|npApplication|) (|npTrap|))
  (|npPush| (funcall fn (|npPop2|) (|npPop1|)))))
```

———-

7.0.414 defun npRestrict

[npTypedForm p218]
[pfRestrict p281]

— **defun npRestrict** —

```
(defun |npRestrict| ()
 (|npTypedForm| 'at #'|pfRestrict|))
```

———-

7.0.415 defun npListofFun

[npTrap p210]
[npPush p141]
[npPop3 p142]
[npPop2 p142]
[npPop1 p142]
[$stack p??]

— **defun npListofFun** —

```
(defun |npListofFun| (f h g)
 (let (a)
 (declare (special |$stack|))
  (cond
   ((apply f nil)
    (cond
     ((and (apply h nil) (or (apply f nil) (|npTrap|)))
       (setq a |$stack|)
       (setq |$stack| nil)
       (do ()
           ((not (and (apply h nil)
                 (or (apply f nil) (|npTrap|))))))
       (setq |$stack| (cons (nreverse |$stack|) a))
       (|npPush| (funcall g (cons (|npPop3|) (cons (|npPop2|) (|npPop1|))))))
      (t t)))
   (t nil))))
```

———-

7.1 Macro handling

7.1.1 defun phMacro

TPDHERE: The pform function has a leading percent sign. fix this

```
carrier[ptree,...] -> carrier[ptree, ptreePremacro,...]
```

[ncEltQ p414]
[ncPutQ p414]
[macroExpanded p220]
[pform p??]

— **defun phMacro** —

```
(defun |phMacro| (carrier)
```

```
(let (ptree)
 (setq ptree (|ncEltQ| carrier '|ptree|))
 (|ncPutQ| carrier '|ptreePremacro| ptree)
 (setq ptree (|macroExpanded| ptree))
 (|ncPutQ| carrier '|ptree| ptree)
 'ok))
```

———-

7.1.2 defun macroExpanded

$macActive is a list of the bodies being expanded. $posActive is a list of the parse forms where the bodies came from. [macExpand p220]
[$posActive p**??**]
[$macActive p**??**]

— **defun macroExpanded** —

```
(defun |macroExpanded| (pf)
 (let (|$posActive| |$macActive|)
 (declare (special |$posActive| |$macActive|))
  (setq |$macActive| nil)
  (setq |$posActive| nil)
  (|macExpand| pf)))
```

———-

7.1.3 defun macExpand

[pfWhere? p292]
[macWhere p226]
[pfLambda? p271]
[macLambda p226]
[pfMacro? p275]
[macMacro p227]
[pfId? p244]
[macId p225]
[pfApplication? p253]
[macApplication p221]
[pfMapParts p234]
[macExpand p220]

— **defun macExpand** —

```
(defun |macExpand| (pf)
 (cond
  ((|pfWhere?| pf)       (|macWhere| pf))
  ((|pfLambda?| pf)      (|macLambda| pf))
  ((|pfMacro?| pf)       (|macMacro| pf))
  ((|pfId?| pf)          (|macId| pf))
  ((|pfApplication?| pf) (|macApplication| pf))
  (t                     (|pfMapParts| #'|macExpand| pf))))
```

———-

7.1.4 defun macApplication

[pfMapParts p234]
[macExpand p220]
[pfApplicationOp p252]
[pfMLambda? p276]
[pf0ApplicationArgs p235]
[mac0MLambdaApply p221]
[$pfMacros p98]

— **defun macApplication** —

```
(defun |macApplication| (pf)
 (let (args op)
 (declare (special |$pfMacros|))
  (setq pf (|pfMapParts| #'|macExpand| pf))
  (setq op (|pfApplicationOp| pf))
  (cond
   ((null (|pfMLambda?| op)) pf)
   (t
    (setq args (|pf0ApplicationArgs| pf))
    (|mac0MLambdaApply| op args pf |$pfMacros|)))))
```

———-

7.1.5 defun mac0MLambdaApply

TPDHERE: The pform function has a leading percent sign. fix this [pf0MLambdaArgs p276]
[pfMLambdaBody p277]
[pfSourcePosition p236]
[ncHardError p350]
[pfId? p244]

[pform p**??**]
[mac0Define p228]
[mac0ExpandBody p222]
[$pfMacros p98]
[$posActive p**??**]
[$macActive p**??**]

— **defun mac0MLambdaApply** —

```
(defun |mac0MLambdaApply| (mlambda args opf |$pfMacros|)
 (declare (special |$pfMacros|))
 (let (pos body params)
 (declare (special |$posActive| |$macActive|))
  (setq params (|pf0MLambdaArgs| mlambda))
  (setq body (|pfMLambdaBody| mlambda))
  (cond
   ((not (eql (length args) (length params)))
    (setq pos (|pfSourcePosition| opf))
    (|ncHardError| pos 'S2CM0003 (list (length params) (length args))))
   (t
    ((lambda (parms p arrgs a) ; for p in params for a in args repeat
     (loop
      (cond
       ((or (atom parms)
            (progn (setq p (car parms)) nil)
            (atom arrgs)
            (progn (setq a (CAR arrgs)) nil))
        (return nil))
       (t
        (cond
         ((null (|pfId?| p))
          (setq pos (|pfSourcePosition| opf))
          (|ncHardError| pos 'S2CM0004 (list (|%pform| p))))
         (t
          (|mac0Define| (|pfIdSymbol| p) '|mparam| a)))))
      (setq parms (cdr parms))
      (setq arrgs (cdr arrgs))))
     params nil args nil)
    (|mac0ExpandBody| body opf |$macActive| |$posActive|)))))
```

———-

7.1.6 defun mac0ExpandBody

[pfSourcePosition p236]
[mac0InfiniteExpansion p223]
[macExpand p220]

[$posActive p??]
[$macActive p??]

— **defun mac0ExpandBody** —

```
(defun |mac0ExpandBody| (body opf |$macActive| |$posActive|)
 (declare (special |$macActive| |$posActive|))
 (let (posn pf)
  (cond
   ((member body |$macActive|)
    (setq pf (cadr |$posActive|))
    (setq posn (|pfSourcePosition| pf))
    (|mac0InfiniteExpansion| posn body |$macActive|))
   (t
    (setq |$macActive| (cons body |$macActive|))
    (setq |$posActive| (cons opf |$posActive|))
    (|macExpand| body)))))
```

———-

7.1.7 defun mac0InfiniteExpansion

TPDHERE: The pform function has a leading percent sign. fix this [mac0InfiniteExpansion,name p224]
[ncSoftError p349]
[pform p??]

— **defun mac0InfiniteExpansion** —

```
(defun |mac0InfiniteExpansion| (posn body active)
 (let (rnames fname tmp1 blist result)
  (setq blist (cons body active))
  (setq tmp1 (mapcar #'|mac0InfiniteExpansion,name| blist))
  (setq fname (car tmp1)) ;[fname, :rnames] := [name b for b in blist]
  (setq rnames (cdr tmp1))
  (|ncSoftError| posn 'S2CM0005
   (list
    (dolist (n (reverse rnames) (nreverse result))
      (setq result (append (reverse (list n "==>")) result)))
    fname (|%pform| body)))
  body))
```

———-

7.1.8 defun mac0InfiniteExpansion,name

[mac0GetName p224]
[pname p1039]

— **defun mac0InfiniteExpansion,name 0** —

```
(defun |mac0InfiniteExpansion,name| (b)
 (let (st sy got)
  (setq got (|mac0GetName| b))
  (cond
   ((null got) "???")
   (t
    (setq sy (car got))
    (setq st (cadr got))
    (if (eq st '|mlambda|)
     (concat (pname sy) "(...)")
     (pname sy))))))
```

———-

7.1.9 defun mac0GetName

Returns [state, body] or NIL. Returns [sy, state] or NIL. [pfMLambdaBody p277]
[$pfMacros p98]

— **defun mac0GetName** —

```
(defun |mac0GetName| (body)
 (let (bd tmp1 st tmp2 sy name)
 (declare (special |$pfMacros|))
 ; for [sy,st,bd] in $pfMacros while not name repeat
  ((lambda (macros tmplist)
   (loop
    (cond
     ((or (atom macros)
          (progn (setq tmplist (car macros)) nil)
          name)
       (return nil))
     (t
      (and (consp tmplist)
           (progn
            (setq sy (car tmplist))
            (setq tmp2 (cdr tmplist))
            (and (consp tmp2)
                 (progn
                  (setq st (car tmp2))
```

```
                (setq tmp1 (cdr tmp2))
                (and (consp tmp1)
                     (eq (cdr tmp1) nil)
                     (progn
                      (setq bd (car tmp1))
                      t)))))
          (progn
           (when (eq st '|mlambda|) (setq bd (|pfMLambdaBody| bd)))
           (when (eq bd body) (setq name (list sy st)))))))
          (setq macros (cdr macros))))
         |$pfMacros| nil)
        name))
```

———-

7.1.10 defun macId

[pfIdSymbol p245]
[mac0Get p226]
[pfCopyWithPos p234]
[pfSourcePosition p236]
[mac0ExpandBody p222]
[$posActive p??]
[$macActive p??]

— **defun macId** —

```
(defun |macId| (pf)
 (let (body state got sy)
 (declare (special |$posActive| |$macActive|))
  (setq sy (|pfIdSymbol| pf))
  (cond
   ((null (setq got (|mac0Get| sy))) pf)
   (t
    (setq state (car got))
    (setq body (cadr got))
    (cond
     ((eq state '|mparam|) body)
     ((eq state '|mlambda|) (|pfCopyWithPos| body (|pfSourcePosition| pf)))
     (t
      (|pfCopyWithPos|
       (|mac0ExpandBody| body pf |$macActive| |$posActive|)
       (|pfSourcePosition| pf))))))))
```

———-

7.1.11 defun mac0Get

[ifcdr p**??**]
[$pfMacros p98]

— **defun mac0Get** —

```
(defun |mac0Get| (sy)
 (declare (special |$pfMacros|))
 (ifcdr (assoc sy |$pfMacros|)))
```

————-

7.1.12 defun macWhere

[macWhere,mac p226]
[$pfMacros p98]

— **defun macWhere** —

```
(defun |macWhere| (pf)
 (declare (special |$pfMacros|))
 (|macWhere,mac| pf |$pfMacros|))
```

————-

7.1.13 defun macWhere,mac

[pfMapParts p234]
[macExpand p220]
[$pfMacros p98]

— **defun macWhere,mac** —

```
(defun |macWhere,mac| (pf |$pfMacros|)
 (declare (special |$pfMacros|))
 (|pfMapParts| #'|macExpand| pf))
```

————-

7.1.14 defun macLambda

[macLambda,mac p227]
[$pfMacros p98]

— **defun macLambda** —

```
(defun |macLambda| (pf)
 (declare (special |$pfMacros|))
 (|macLambda,mac| pf |$pfMacros|))
```

———-

7.1.15 defun macLambda,mac

[pfMapParts p234]
[macExpand p220]
[$pfMacros p98]

— **defun macLambda,mac** —

```
(defun |macLambda,mac| (pf |$pfMacros|)
 (declare (special |$pfMacros|))
 (|pfMapParts| #'|macExpand| pf))
```

———-

7.1.16 defun Add appropriate definition the a Macro pform

This function adds the appropriate definition and returns the original Macro pform. **TPDHERE: The pform function has a leading percent sign. fix this** [pfMacroLhs p275]
[pfMacroRhs p275]
[pfId? p244]
[ncSoftError p349]
[pfSourcePosition p236]
[pfIdSymbol p245]
[mac0Define p228]
[pform p**??**]
[pfMLambda? p276]
[macSubstituteOuter p228]
[pfNothing? p243]
[pfMacro p275]
[pfNothing p243]

— **defun macMacro** —

```
(defun |macMacro| (pf)
 (let (sy rhs lhs)
```

```
  (setq lhs (|pfMacroLhs| pf))
  (setq rhs (|pfMacroRhs| pf))
  (cond
   ((null (|pfId?| lhs))
    (|ncSoftError| (|pfSourcePosition| lhs) 'S2CM0001 (list (|%pform| lhs)))
    pf)
   (t
    (setq sy (|pfIdSymbol| lhs))
    (|mac0Define| sy
     (cond
      ((|pfMLambda?| rhs) '|mlambda|)
      (t '|mbody|))
     (|macSubstituteOuter| rhs))
     (cond
      ((|pfNothing?| rhs) pf)
      (t (|pfMacro| lhs (|pfNothing|)))))))))
```

———-

7.1.17 defun Add a macro to the global pfMacros list

[$pfMacros p98]

— **defun mac0Define 0** —

```
(defun |mac0Define| (sy state body)
 (declare (special |$pfMacros|))
  (setq |$pfMacros| (cons (list sy state body) |$pfMacros|)))
```

———-

7.1.18 defun macSubstituteOuter

[mac0SubstituteOuter p229]
[macLambdaParameterHandling p229]

— **defun macSubstituteOuter** —

```
(defun |macSubstituteOuter| (pform)
 (|mac0SubstituteOuter| (|macLambdaParameterHandling| nil pform) pform))
```

———-

7.1.19 defun mac0SubstituteOuter

[pfId? p244]
[macSubstituteId p230]
[pfLeaf? p245]
[pfLambda? p271]
[macLambdaParameterHandling p229]
[mac0SubstituteOuter p229]
[pfParts p247]

— **defun mac0SubstituteOuter** —

```
(defun |mac0SubstituteOuter| (replist pform)
 (let (tmplist)
  (cond
   ((|pfId?| pform) (|macSubstituteId| replist pform))
   ((|pfLeaf?| pform) pform)
   ((|pfLambda?| pform)
    (setq tmplist (|macLambdaParameterHandling| replist pform))
    (dolist (p (|pfParts| pform)) (|mac0SubstituteOuter| tmplist p))
    pform)
   (t
    (dolist (p (|pfParts| pform)) (|mac0SubstituteOuter| replist p))
    pform))))
```

———-

7.1.20 defun macLambdaParameterHandling

[pfLeaf? p245]
[pfLambda? p271]
[pfTypedId p289]
[pf0LambdaArgs p272]
[pfIdSymbol p245]
[pfMLambda? p276]
[pf0MLambdaArgs p276]
[pfLeaf p245]
[pfAbSynOp p410]
[pfLeafPosition p246]
[pfParts p247]
[macLambdaParameterHandling p229]

— **defun macLambdaParameterHandling** —

```
(defun |macLambdaParameterHandling| (replist pform)
 (let (parlist symlist result)
```

```
  (cond
   ((|pfLeaf?| pform) nil)
   ((|pfLambda?| pform) ; remove ( identifier . replacement ) from assoclist
    (setq parlist (mapcar #'|pfTypedId| (|pf0LambdaArgs| pform)))
    (setq symlist (mapcar #'|pfIdSymbol| parlist))
    (dolist (par symlist)
     (setq replist
      (let ((pr (assoc par replist :test #'equal)))
       (when pr (remove par replist :test #'equal)))))
    replist)
   ((|pfMLambda?| pform) ;construct assoclist ( identifier . replacement )
    (setq parlist (|pf0MLambdaArgs| pform)) ; extract parameter list
    (dolist (par parlist (nreverse result))
     (push
      (cons (|pfIdSymbol| par)
            (|pfLeaf| (|pfAbSynOp| par) (gensym) (|pfLeafPosition| par)))
       result)))
   (t
    (dolist (p (|pfParts| pform))
     (|macLambdaParameterHandling| replist p))))))
```

———-

7.1.21 defun macSubstituteId

[pfIdSymbol p245]

— **defun macSubstituteId** —

```
(defun |macSubstituteId| (replist pform)
 (let (ex)
  (setq ex (assoc (|pfIdSymbol| pform) replist :test #'eq))
  (cond
    (ex
     (rplaca pform (cadr ex))
     (rplacd pform (cddr ex))
     pform)
    (t pform))))
```

———-

Chapter 8

Pftrees

8.1 Abstract Syntax Trees Overview

Th functions create and examine abstract syntax trees. These are called pforms, for short.
The pform data structure

- Leaves: [hd, tok, pos] where pos is optional
- Trees: [hd, tree, tree, ...]
- hd is either an id or (id . alist)

The leaves are:

char	:=	('char expr position)
Document	:=	('Document expr position)
error	:=	('error expr position)
expression	:=	('expression expr position)
float	:=	('float expr position)
id	:=	('id expr position)
idsy	:=	('idsy expr position)
integer	:=	('integer expr position)
string	:=	('string expr position)
symbol	:=	('symbol expr position)

The special nodes:

ListOf	:=	('listOf items)
Nothing	:=	('nothing)
SemiColon	:=	('SemiColon (Body: Expr))

The expression nodes:

```
Add                 :=  ('Add (Base: [Typed], Addin: Expr))
And                 :=  ('And left right)
Application         :=  ('Application (Op: Expr, Arg: Expr))
Assign              :=  ('Assign (LhsItems: [AssLhs], Rhs: Expr))
Attribute           :=  ('Attribute (Expr: Primary))
Break               :=  ('Break (From: ? Id))
Coerceto            :=  ('Coerceto (Expr: Expr, Type: Type))
Collect             :=  ('Collect (Body: Expr, Iterators: [Iterator]))
ComDefinition       :=  ('ComDefinition (Doc: Document, Def: Definition))
DeclPart
Definition          :=  ('Definition (LhsItems: [Typed], Rhs: Expr))
DefinitionSequence  :=  (Args: [DeclPart])
Do                  :=  ('Do (Body: Expr))
Document            :=  ('Document strings)
DWhere              :=  ('DWhere (Context: [DeclPart], Expr: [DeclPart]))
EnSequence          :=
Exit                :=  ('Exit (Cond: ? Expr, Expr: ? Expr))
Export              :=  ('Export (Items: [Typed]))
Forin               :=  ('Forin (Lhs: [AssLhs], Whole: Expr))
Free                :=  ('Free (Items: [Typed]))
Fromdom             :=  ('Fromdom (What: Id, Domain: Type))
Hide                :=  ('hide, arg)
If                  :=  ('If (Cond: Expr, Then: Expr, Else: ? Expr))
Import              :=  ('Import (Items: [QualType]))
Inline              :=  ('Inline (Items: [QualType]))
Iterate             :=  ('Iterate (From: ? Id))
Lambda              :=  ('Lambda (Args: [Typed], Rets: ReturnedTyped, Body: Expr))
Literal
Local               :=  ('Local (Items: [Typed]))
Loop                :=  ('Loop (Iterators: [Iterator]))
Macro               :=  ('Macro (Lhs: Id, Rhs: ExprorNot))
MLambda             :=  ('MLambda (Args: [Id], Body: Expr))
Not                 :=  ('Not arg)
Novalue             :=  ('Novalue (Expr: Expr))
Or                  :=  ('Or left right)
Pretend             :=  ('Pretend (Expr: Expr, Type: Type))
QualType            :=  ('QualType (Type: Type, Qual: ? Type))
Restrict            :=  ('Restrict (Expr: Expr, Type: Type))
Retract             :=  ('RetractTo (Expr: Expr, Type: Type))
Return              :=  ('Return (Expr: ? Expr, From: ? Id))
ReturnTyped         :=  ('returntyuped (type body))
Rule                :=  ('Rule (lhsitems, rhsitems))
Sequence            :=  ('Sequence (Args: [Expr]))
Suchthat            :=  ('Suchthat (Cond: Expr))
Symb                :=  if leaf then symbol else expression
Tagged              :=  ('Tagged (Tag: Expr, Expr: Expr))
TLambda             :=  ('TLambda (Args: [Typed],
                          Rets: ReturnedTyped Type, Body: Expr))
Tuple               :=  ('Tuple (Parts: [Expr]))
Typed               :=  ('Typed (Id: Id, Type: ? Type))
Typing              :=  ('Typing (Items: [Typed]))
Until               :=  ('Until (Cond: Expr)) NOT USED
WDeclare            :=  ('WDeclare (Signature: Typed, Doc: ? Document))
Where               :=  ('Where (Context: [DeclPart], Expr: Expr))
While               :=  ('While (Cond: Expr))
With                :=  ('With (Base: [Typed], Within: [WithPart]))
```

Special cases of expression nodes are:

- Application. The Op parameter is one of `and, or, Y, |, {}, [], {||}, [||]`
- DeclPart. The comment is attached to all signatutres in Typing, Import, Definition, Sequence, DWhere, Macro nodes
- EnSequence. This is either a Tuple or Sequence depending on the argument
- Literal. One of integer symbol expression one zero char string float of the form ('expression expr position)

8.2 Structure handlers

8.2.1 defun pfGlobalLinePosn

[poGlobalLinePosn p72]

— **defun pfGlobalLinePosn** —

```
(defun |pfGlobalLinePosn| (posn)
 (|poGlobalLinePosn| posn))
```

————-

8.2.2 defun pfCharPosn

[poCharPosn p375]

— **defun pfCharPosn** —

```
(defun |pfCharPosn| (posn)
 (|poCharPosn| posn))
```

————-

8.2.3 defun pfLinePosn

[poLinePosn p359]

— **defun pfLinePosn** —

```
(defun |pfLinePosn| (posn)
 (|poLinePosn| posn))
```

———-

8.2.4 defun pfFileName

[poFileName p358]

— **defun pfFileName** —

```
(defun |pfFileName| (posn)
 (|poFileName| posn))
```

———-

8.2.5 defun pfCopyWithPos

[pfLeaf? p245]
[pfLeaf p245]
[pfAbSynOp p410]
[tokPart p411]
[pfTree p250]
[pfParts p247]
[pfCopyWithPos p234]

— **defun pfCopyWithPos** —

```
(defun |pfCopyWithPos| (pform pos)
 (if (|pfLeaf?| pform)
  (|pfLeaf| (|pfAbSynOp| pform) (|tokPart| pform) pos)
  (|pfTree| (|pfAbSynOp| pform)
            (loop for p in (|pfParts| pform)
                  collect (|pfCopyWithPos| p pos)))))
```

———-

8.2.6 defun pfMapParts

[pfLeaf? p245]
[pfParts p247]
[pfTree p250]

[pfAbSynOp p410]

— **defun pfMapParts** —

```
(defun |pfMapParts| (f pform)
 (let (parts1 parts0)
  (if (|pfLeaf?| pform)
   pform
   (progn
    (setq parts0 (|pfParts| pform))
    (setq parts1 (loop for p in parts0 collect (funcall f p)))
    (if (reduce #'(lambda (u v) (and u v)) (mapcar #'eq parts0 parts1))
     pform
     (|pfTree| (|pfAbSynOp| pform) parts1))))))
```

————-

8.2.7 defun pf0ApplicationArgs

[pf0FlattenSyntacticTuple p235]
[pfApplicationArg p252]

— **defun pf0ApplicationArgs** —

```
(defun |pf0ApplicationArgs| (pform)
  (|pf0FlattenSyntacticTuple| (|pfApplicationArg| pform)))
```

————-

8.2.8 defun pf0FlattenSyntacticTuple

[pfTuple? p290]
[pf0FlattenSyntacticTuple p235]
[pf0TupleParts p291]

— **defun pf0FlattenSyntacticTuple** —

```
(defun |pf0FlattenSyntacticTuple| (pform)
 (if (null (|pfTuple?| pform))
  (list pform)
   ; [:pf0FlattenSyntacticTuple p for p in pf0TupleParts pform]
  ((lambda (arg0 arg1 p)
   (loop
    (cond
     ((or (atom arg1) (progn (setq p (car arg1)) nil))
```

```
      (return (nreverse arg0)))
     (t
      (setq arg0 (append (reverse (|pf0FlattenSyntacticTuple| p)) arg0))))
    (setq arg1 (cdr arg1))))
   nil (|pf0TupleParts| pform) nil)))
```

———-

8.2.9 defun pfSourcePosition

[pfLeaf? p245]
[pfLeafPosition p246]
[poNoPosition? p411]
[pfSourcePosition p236]
[pfParts p247]
[$nopos p28]

— **defun pfSourcePosition** —

```
(defun |pfSourcePosition| (form)
 (let (pos)
 (declare (special |$nopos|))
  (cond
   ((|pfLeaf?| form) (|pfLeafPosition| form))
   (t
    (setq pos |$nopos|)
    ((lambda (theparts p) ; for p in parts while poNoPosition? pos repeat
     (loop
      (cond
       ((or (atom theparts)
            (progn (setq p (car theparts)) nil)
            (not (|poNoPosition?| pos)))
          (return nil))
       (t (setq pos (|pfSourcePosition| p))))
      (setq theparts (cdr theparts))))
     (|pfParts| form) nil)
    pos))))
```

———-

8.2.10 defun Convert a Sequence node to a list

[pfSequence? p285]
[pfSequenceArgs p285]
[pfListOf p243]

— **defun pfSequenceToList** —

```
(defun |pfSequenceToList| (x)
 (if (|pfSequence?| x)
  (|pfSequenceArgs| x)
  (|pfListOf| (list x))))
```

————-

8.2.11 defun pfSpread

[pfTyped p288]

— **defun pfSpread** —

```
(defun |pfSpread| (arg1 arg2)
 (mapcar #'(lambda (i) (|pfTyped| i arg2)) arg1))
```

————-

8.2.12 defun Deconstruct nodes to lists

[pfTagged? p287]
[pfTaggedExpr p287]
[pfNothing p243]
[pfTaggedTag p287]
[pfId? p244]
[pfListOf p243]
[pfTyped p288]
[pfCollect1? p240]
[pfCollectVariable1 p240]
[pfTuple? p290]
[pf0TupleParts p291]
[pfTaggedToTyped p287]
[pfDefinition? p259]
[pfApplication? p253]
[pfFlattenApp p239]
[pfTaggedToTyped1 p242]
[pfTransformArg p242]
[npTrapForm p210]

— **defun pfCheckItOut** —

```
(defun |pfCheckItOut| (x)
 (let (args op ls form rt result)
  (if (|pfTagged?| x)
   (setq rt (|pfTaggedExpr| x))
   (setq rt (|pfNothing|)))
  (if (|pfTagged?| x)
   (setq form (|pfTaggedTag| x))
   (setq form x))
  (cond
   ((|pfId?| form)
    (list (|pfListOf| (list (|pfTyped| form rt))) nil rt))
   ((|pfCollect1?| form)
    (list (|pfListOf| (list (|pfCollectVariable1| form))) nil rt))
   ((|pfTuple?| form)
    (list (|pfListOf|
     (dolist (part (|pf0TupleParts| form) (nreverse result))
      (push (|pfTaggedToTyped| part) result)))
     nil rt))
   ((|pfDefinition?| form)
    (list (|pfListOf| (list (|pfTyped| form (|pfNothing|)))) nil rt))
   ((|pfApplication?| form)
    (setq ls (|pfFlattenApp| form))
    (setq op (|pfTaggedToTyped1| (car ls)))
    (setq args
     (dolist (part (cdr ls) (nreverse result))
      (push (|pfTransformArg| part) result)))
    (list (|pfListOf| (list op)) args rt))
   (t (|npTrapForm| form)))))
```

———-

8.2.13 defun pfCheckMacroOut

[pfId? p244]
[pfApplication? p253]
[pfFlattenApp p239]
[pfCheckId p239]
[pfCheckArg p239]
[npTrapForm p210]

— defun pfCheckMacroOut —

```
(defun |pfCheckMacroOut| (form)
 (let (args op ls)
  (cond
   ((|pfId?| form) (list form nil))
   ((|pfApplication?| form)
```

```
    (setq ls (|pfFlattenApp| form))
    (setq op (|pfCheckId| (car ls)))
    (setq args (mapcar #'|pfCheckArg| (cdr ls)))
    (list op args))
   (t (|npTrapForm| form)))))
```

———-

8.2.14 defun pfCheckArg

[pfTuple? p290]
[pf0TupleParts p291]
[pfListOf p243]
[pfCheckId p239]

— **defun pfCheckArg** —

```
(defun |pfCheckArg| (args)
 (let (argl)
  (if (|pfTuple?| args)
   (setq argl (|pf0TupleParts| args))
   (setq argl (list args)))
  (|pfListOf| (mapcar #'|pfCheckId| argl))))
```

———-

8.2.15 defun pfCheckId

[pfId? p244]
[npTrapForm p210]

— **defun pfCheckId** —

```
(defun |pfCheckId| (form)
 (if (null (|pfId?| form))
  (|npTrapForm| form)
  form))
```

———-

8.2.16 defun pfFlattenApp

[pfApplication? p253]
[pfCollect1? p240]

[pfFlattenApp p239]
[pfApplicationOp p252]
[pfApplicationArg p252]

— **defun pfFlattenApp** —

```
(defun |pfFlattenApp| (x)
 (cond
  ((|pfApplication?| x)
   (cond
    ((|pfCollect1?| x) (LIST x))
    (t
     (append (|pfFlattenApp| (|pfApplicationOp| x))
             (|pfFlattenApp| (|pfApplicationArg| x))))))
  (t (list x))))
```

———-

8.2.17 defun pfCollect1?

[pfApplication? p253]
[pfApplicationOp p252]
[pfId? p244]
[pfIdSymbol p245]

— **defun pfCollect1?** —

```
(defun |pfCollect1?| (x)
 (let (a)
  (when (|pfApplication?| x)
   (setq a (|pfApplicationOp| x))
   (when (|pfId?| a) (eq (|pfIdSymbol| a) '|\||)))))
```

———-

8.2.18 defun pfCollectVariable1

[pfApplicationArg p252]
[pf0TupleParts p291]
[pfTaggedToTyped p287]
[pfTyped p288]
[pfSuch p242]
[pfTypedId p289]
[pfTypedType p289]

— **defun pfCollectVariable1** —

```
(defun |pfCollectVariable1| (x)
 (let (id var a)
  (setq a (|pfApplicationArg| x))
  (setq var (car (|pf0TupleParts| a)))
  (setq id (|pfTaggedToTyped| var))
  (|pfTyped|
   (|pfSuch| (|pfTypedId| id) (cadr (|pf0TupleParts| a)))
   (|pfTypedType| id))))
```

————-

8.2.19 defun pfPushMacroBody

[pfMLambda p276]
[pfPushMacroBody p241]

— **defun pfPushMacroBody** —

```
(defun |pfPushMacroBody| (args body)
 (if (null args)
   body
   (|pfMLambda| (car args) (|pfPushMacroBody| (cdr args) body))))
```

————-

8.2.20 defun pfSourceStok

[pfLeaf? p245]
[pfParts p247]
[pfSourceStok p241]
[pfFirst p262]

— **defun pfSourceStok** —

```
(defun |pfSourceStok| (x)
 (cond
  ((|pfLeaf?| x) x)
  ((null (|pfParts| x)) '|NoToken|)
  (t (|pfSourceStok| (|pfFirst| x)))))
```

————-

8.2.21 defun pfTransformArg

[pfTuple? p290]
[pf0TupleParts p291]
[pfListOf p243]
[pfTaggedToTyped1 p242]

— **defun pfTransformArg** —

```
(defun |pfTransformArg| (args)
 (let (arglist result)
  (if (|pfTuple?| args)
   (setq arglist (|pf0TupleParts| args))
   (setq arglist (list args)))
  (|pfListOf|
   (dolist (|i| arglist (nreverse result))
    (push (|pfTaggedToTyped1| |i|) result)))))
```

———-

8.2.22 defun pfTaggedToTyped1

[pfCollect1? p240]
[pfCollectVariable1 p240]
[pfDefinition? p259]
[pfTyped p288]
[pfNothing p243]
[pfTaggedToTyped p287]

— **defun pfTaggedToTyped1** —

```
(defun |pfTaggedToTyped1| (arg)
 (cond
  ((|pfCollect1?| arg) (|pfCollectVariable1| arg))
  ((|pfDefinition?| arg) (|pfTyped| arg (|pfNothing|)))
  (t (|pfTaggedToTyped| arg))))
```

———-

8.2.23 defun pfSuch

[pfInfApplication p269]
[pfId p244]

— **defun pfSuch** —

```
(defun |pfSuch| (x y)
 (|pfInfApplication| (|pfId| '|\||) x y))
```

———-

8.3 Special Nodes

8.3.1 defun Create a Listof node

[pfTree p250]

— **defun pfListOf** —

```
(defun |pfListOf| (x)
 (|pfTree| '|listOf| x))
```

———-

8.3.2 defun pfNothing

[pfTree p250]

— **defun pfNothing** —

```
(defun |pfNothing| ()
 (|pfTree| '|nothing| nil))
```

———-

8.3.3 defun Is this a Nothing node?

[pfAbSynOp? p410]

— **defun pfNothing?** —

```
(defun |pfNothing?| (form)
 (|pfAbSynOp?| form '|nothing|))
```

———-

8.4 Leaves

8.4.1 defun Create a Document node

[pfLeaf p245]

— **defun pfDocument** —

```
(defun |pfDocument| (strings)
 (|pfLeaf| '|Document| strings))
```

————-

8.4.2 defun Construct an Id node

[pfLeaf p245]

— **defun pfId** —

```
(defun |pfId| (expr)
 (|pfLeaf| '|id| expr))
```

————-

8.4.3 defun Is this an Id node?

[pfAbSynOp? p410]

— **defun pfId?** —

```
(defun |pfId?| (form)
 (or (|pfAbSynOp?| form '|id|) (|pfAbSynOp?| form '|idsy|)))
```

————-

8.4.4 defun Construct an Id leaf node

[pfLeaf p245]

— **defun pfIdPos** —

```
(defun |pfIdPos| (expr pos)
 (|pfLeaf| '|id| expr pos))
```

———-

8.4.5 defun Return the Id part

[tokPart p411]

— **defun pfIdSymbol** —

```
(defun |pfIdSymbol| (form)
 (|tokPart| form))
```

———-

8.4.6 defun Construct a Leaf node

[tokConstruct p409]
[ifcar p**??**]
[pfNoPosition p412]

— **defun pfLeaf** —

```
(defun |pfLeaf| (x y &rest z)
 (|tokConstruct| x y (or (ifcar z) (|pfNoPosition|))))
```

———-

8.4.7 defun Is this a leaf node?

[pfAbSynOp p410]

— **defun pfLeaf?** —

```
(defun |pfLeaf?| (form)
 (member (|pfAbSynOp| form)
   '(|id| |idsy| |symbol| |string| |char| |float| |expression|
     |integer| |Document| |error|)))
```

———-

8.4.8 defun Return the token position of a leaf node

[tokPosn p411]

— **defun pfLeafPosition** —

```
(defun |pfLeafPosition| (form)
 (|tokPosn| form))
```

————-

8.4.9 defun Return the Leaf Token

[tokPart p411]

— **defun pfLeafToken** —

```
(defun |pfLeafToken| (form)
 (|tokPart| form))
```

————-

8.4.10 defun Is this a Literal node?

[pfAbSynOp p410]

— **defun pfLiteral? 0** —

```
(defun |pfLiteral?| (form)
 (member (|pfAbSynOp| form)
  '(|integer| |symbol| |expression| |one| |zero| |char| |string| |float|)))
```

————-

8.4.11 defun Create a LiteralClass node

[pfAbSynOp p410]

— **defun pfLiteralClass** —

```
(defun |pfLiteralClass| (form)
 (|pfAbSynOp| form))
```

————-

8.4.12 defun Return the LiteralString

[tokPart p411]

— **defun pfLiteralString** —

```
(defun |pfLiteralString| (form)
 (|tokPart| form))
```

————-

8.4.13 defun Return the parts of a tree node

— **defun pfParts 0** —

```
(defun |pfParts| (form)
 (cdr form))
```

————-

8.4.14 defun Return the argument unchanged

— **defun pfPile 0** —

```
(defun |pfPile| (part)
 part)
```

————-

8.4.15 defun pfPushBody

[pfLambda p271]
[pfNothing p243]
[pfPushBody p247]

— **defun pfPushBody** —

```
(defun |pfPushBody| (rt args body)
 (cond
  ((null args) body)
```

```
  ((null (cdr args)) (|pfLambda| (car args) rt body))
  (t
   (|pfLambda| (car args) (|pfNothing|)
               (|pfPushBody| rt (cdr args) body)))))
```

———-

8.4.16 defun An S-expression which people can read.

[pfSexpr,strip p248]

— **defun pfSexpr** —

```
(defun |pfSexpr| (pform)
 (|pfSexpr,strip| pform))
```

———-

8.4.17 defun Create a human readable S-expression

[pfId? p244]
[pfIdSymbol p245]
[pfLiteral? p246]
[pfLiteralString p247]
[pfLeaf? p245]
[tokPart p411]
[pfApplication? p253]
[pfApplicationArg p252]
[pfTuple? p290]
[pf0TupleParts p291]
[pfApplicationOp p252]
[pfSexpr,strip p248]
[pfAbSynOp p410]
[pfParts p247]

— **defun pfSexpr,strip** —

```
(defun |pfSexpr,strip| (pform)
 (let (args a result)
  (cond
   ((|pfId?| pform)      (|pfIdSymbol| pform))
   ((|pfLiteral?| pform) (|pfLiteralString| pform))
   ((|pfLeaf?| pform)    (|tokPart| pform))
   ((|pfApplication?| pform)
```

```
    (setq a (|pfApplicationArg| pform))
    (if (|pfTuple?| a)
     (setq args (|pf0TupleParts| a))
     (setq args (list a)))
    (dolist (p (cons (|pfApplicationOp| pform) args) (nreverse result))
     (push (|pfSexpr,strip| p) result)))
   (t
    (cons (|pfAbSynOp| pform)
     (dolist (p (|pfParts| pform) (nreverse result))
      (push (|pfSexpr,strip| p) result)))))))
```

———-

8.4.18 defun Construct a Symbol or Expression node

[pfLeaf? p245]
[pfSymbol p249]
[tokPart p411]
[ifcar p**??**]
[pfExpression p262]
[pfSexpr p248]

— **defun pfSymb** —

```
(defun |pfSymb| (expr &REST optpos)
 (if (|pfLeaf?| expr)
  (|pfSymbol| (|tokPart| expr) (ifcar optpos))
  (|pfExpression| (|pfSexpr| expr) (ifcar optpos))))
```

———-

8.4.19 defun Construct a Symbol leaf node

[pfLeaf p245]
[ifcar p**??**]

— **defun pfSymbol** —

```
(defun |pfSymbol| (expr &rest optpos)
 (|pfLeaf| '|symbol| expr (ifcar optpos)))
```

———-

8.4.20 defun Is this a Symbol node?

[pfAbSynOp? p410]

— **defun pfSymbol?** —

```
(defun |pfSymbol?| (form)
 (|pfAbSynOp?| form '|symbol|))
```

———-

8.4.21 defun Return the Symbol part

[tokPart p411]

— **defun pfSymbolSymbol** —

```
(defun |pfSymbolSymbol| (form)
 (|tokPart| form))
```

———-

8.5 Trees

8.5.1 defun Construct a tree node

— **defun pfTree 0** —

```
(defun |pfTree| (x y)
 (cons x y)))
```

———-

8.5.2 defun Construct an Add node

[pfNothing p243]
[pfTree p250]

— **defun pfAdd** —

```
(defun |pfAdd| (pfbase pfaddin &rest addon)
 (let (lhs)
  (if addon
   (setq lhs addon)
   (setq lhs (|pfNothing|)))
  (|pfTree| '|Add| (list pfbase pfaddin lhs))))
```

———-

8.5.3 defun Construct an And node

[pfTree p250]

— **defun pfAnd** —

```
(defun |pfAnd| (pfleft pfright)
 (|pfTree| '|And| (list pfleft pfright)))
```

———-

8.5.4 defun pfAttribute

[pfTree p250]

— **defun pfAttribute** —

```
(defun |pfAttribute| (pfexpr)
 (|pfTree| '|Attribute| (list pfexpr)))
```

———-

8.5.5 defun Return an Application node

[pfTree p250]

— **defun pfApplication** —

```
(defun |pfApplication| (pfop pfarg)
 (|pfTree| '|Application| (list pfop pfarg)))
```

———-

8.5.6 defun Return the Arg part of an Application node

— **defun pfApplicationArg 0** —

```
(defun |pfApplicationArg| (pf)
 (caddr pf))
```

————-

8.5.7 defun Return the Op part of an Application node

— **defun pfApplicationOp 0** —

```
(defun |pfApplicationOp| (pf)
 (cadr pf))
```

————-

8.5.8 defun Is this an And node?

[pfAbSynOp? p410]

— **defun pfAnd?** —

```
(defun |pfAnd?| (pf)
 (|pfAbSynOp?| pf '|And|))
```

————-

8.5.9 defun Return the Left part of an And node

— **defun pfAndLeft 0** —

```
(defun |pfAndLeft| (pf)
 (cadr pf))
```

————-

8.5.10 defun Return the Right part of an And node

— **defun pfAndRight 0** —

```
(defun |pfAndRight| (pf)
 (caddr pf))
```

————-

8.5.11 defun Flatten a list of lists

— **defun pfAppend 0** —

```
(defun |pfAppend| (list)
 (apply #'append list))
```

————-

8.5.12 defun Is this an Application node?

[pfAbSynOp? p410]

— **defun pfApplication?** —

```
(defun |pfApplication?| (pf)
  (|pfAbSynOp?| pf '|Application|))
```

————-

8.5.13 defun Create an Assign node

[pfTree p250]

— **defun pfAssign** —

```
(defun |pfAssign| (pflhsitems pfrhs)
 (|pfTree| '|Assign| (list pflhsitems pfrhs)))
```

————-

8.5.14 defun Is this an Assign node?

[pfAbSynOp? p410]

— **defun pfAssign?** —

```
(defun |pfAssign?| (pf)
 (|pfAbSynOp?| pf '|Assign|))
```

———-

8.5.15 defun Return the parts of an LhsItem of an Assign node

[pfParts p247]
[pfAssignLhsItems p254]

— **defun pf0AssignLhsItems 0** —

```
(defun |pf0AssignLhsItems| (pf)
 (|pfParts| (|pfAssignLhsItems| pf)))
```

———-

8.5.16 defun Return the LhsItem of an Assign node

— **defun pfAssignLhsItems 0** —

```
(defun |pfAssignLhsItems| (pf)
 (cadr pf))
```

———-

8.5.17 defun Return the RHS of an Assign node

— **defun pfAssignRhs 0** —

```
(defun |pfAssignRhs| (pf)
 (caddr pf))
```

———-

8.5.18 defun Construct an application node for a brace

[pfApplication p251]
[pfIdPos p244]
[tokPosn p411]

— **defun pfBrace** —

```
(defun |pfBrace| (a part)
 (|pfApplication| (|pfIdPos| '{} (|tokPosn| a)) part))
```

————-

8.5.19 defun Construct an Application node for brace-bars

[pfApplication p251]
[pfIdPos p244]
[tokPosn p411]

— **defun pfBraceBar** —

```
(defun |pfBraceBar| (a part)
 (|pfApplication| (|pfIdPos| '|{\|\|}| (|tokPosn| a)) part))
```

————-

8.5.20 defun Construct an Application node for a bracket

[pfApplication p251]
[pfIdPos p244]
[tokPosn p411]

— **defun pfBracket** —

```
(defun |pfBracket| (a part)
 (|pfApplication| (|pfIdPos| '[] (|tokPosn| a)) part))
```

————-

8.5.21 defun Construct an Application node for bracket-bars

[pfApplication p251]
[pfIdPos p244]

[tokPosn p411]

— **defun pfBracketBar** —

```
(defun |pfBracketBar| (a part)
 (|pfApplication| (|pfIdPos| '|[\|\|]| (|tokPosn| a)) part))
```

———-

8.5.22 defun Create a Break node

[pfTree p250]

— **defun pfBreak** —

```
(defun |pfBreak| (pffrom)
 (|pfTree| '|Break| (list pffrom)))
```

———-

8.5.23 defun Is this a Break node?

[pfAbSynOp? p410]

— **defun pfBreak?** —

```
(defun |pfBreak?| (pf)
 (|pfAbSynOp?| pf '|Break|))
```

———-

8.5.24 defun Return the From part of a Break node

— **defun pfBreakFrom 0** —

```
(defun |pfBreakFrom| (pf)
 (cadr pf))
```

———-

8.5.25 defun Construct a Coerceto node

[pfTree p250]

— **defun pfCoerceto** —

```
(defun |pfCoerceto| (pfexpr pftype)
 (|pfTree| '|Coerceto| (list pfexpr pftype)))
```

———-

8.5.26 defun Is this a CoerceTo node?

[pfAbSynOp? p410]

— **defun pfCoerceto?** —

```
(defun |pfCoerceto?| (pf)
 (|pfAbSynOp?| pf '|Coerceto|))
```

———-

8.5.27 defun Return the Expression part of a CoerceTo node

— **defun pfCoercetoExpr 0** —

```
(defun |pfCoercetoExpr| (pf)
 (cadr pf))
```

———-

8.5.28 defun Return the Type part of a CoerceTo node

— **defun pfCoercetoType 0** —

```
(defun |pfCoercetoType| (pf)
 (caddr pf))
```

———-

8.5.29 defun Return the Body of a Collect node

— **defun pfCollectBody 0** —

```
(defun |pfCollectBody| (pf)
 (cadr pf))
```

————-

8.5.30 defun Return the Iterators of a Collect node

— **defun pfCollectIterators 0** —

```
(defun |pfCollectIterators| (pf)
 (caddr pf))
```

————-

8.5.31 defun Create a Collect node

[pfTree p250]

— **defun pfCollect** —

```
(defun |pfCollect| (pfbody pfiterators)
 (|pfTree| '|Collect| (list pfbody pfiterators)))
```

————-

8.5.32 defun Is this a Collect node?

[pfAbSynOp? p410]

— **defun pfCollect?** —

```
(defun |pfCollect?| (pf)
 (|pfAbSynOp?| pf '|Collect|))
```

————-

8.5.33 defun pfDefinition

[pfTree p250]

— **defun pfDefinition** —

```
(defun |pfDefinition| (pflhsitems pfrhs)
 (|pfTree| '|Definition| (list pflhsitems pfrhs)))
```

———-

8.5.34 defun Return the Lhs of a Definition node

— **defun pfDefinitionLhsItems 0** —

```
(defun |pfDefinitionLhsItems| (pf)
 (cadr pf))
```

———-

8.5.35 defun Return the Rhs of a Definition node

— **defun pfDefinitionRhs 0** —

```
(defun |pfDefinitionRhs| (pf)
 (caddr pf))
```

———-

8.5.36 defun Is this a Definition node?

[pfAbSynOp? p410]

— **defun pfDefinition?** —

```
(defun |pfDefinition?| (pf)
 (|pfAbSynOp?| pf '|Definition|))
```

———-

8.5.37 defun Return the parts of a Definition node

[pfParts p247]
[pfDefinitionLhsItems p259]

— **defun pf0DefinitionLhsItems** —

```
(defun |pf0DefinitionLhsItems| (pf)
 (|pfParts| (|pfDefinitionLhsItems| pf)))
```

———-

8.5.38 defun Create a Do node

[pfTree p250]

— **defun pfDo** —

```
(defun |pfDo| (pfbody)
 (|pfTree| '|Do| (list pfbody)))
```

———-

8.5.39 defun Is this a Do node?

[pfAbSynOp? p410]

— **defun pfDo?** —

```
(defun |pfDo?| (pf)
 (|pfAbSynOp?| pf '|Do|))
```

———-

8.5.40 defun Return the Body of a Do node

— **defun pfDoBody 0** —

```
(defun |pfDoBody| (pf)
 (cadr pf))
```

———-

8.5.41 defun Construct a Sequence node

[pfTuple p290]
[pfListOf p243]
[pfSequence p285]

— **defun pfEnSequence** —

```
(defun |pfEnSequence| (a)
 (cond
  ((null a) (|pfTuple| (|pfListOf| a)))
  ((null (cdr a)) (car a))
  (t (|pfSequence| (|pfListOf| a)))))
```

———-

8.5.42 defun Construct an Exit node

[pfTree p250]

— **defun pfExit** —

```
(defun |pfExit| (pfcond pfexpr)
 (|pfTree| '|Exit| (list pfcond pfexpr)))
```

———-

8.5.43 defun Is this an Exit node?

[pfAbSynOp? p410]

— **defun pfExit?** —

```
(defun |pfExit?| (pf)
 (|pfAbSynOp?| pf '|Exit|))
```

———-

8.5.44 defun Return the Cond part of an Exit

— **defun pfExitCond 0** —

```
(defun |pfExitCond| (pf)
 (cadr pf))
```

———-

8.5.45 defun Return the Expression part of an Exit

— **defun pfExitExpr 0** —

```
(defun |pfExitExpr| (pf)
 (caddr pf))
```

———-

8.5.46 defun Create an Export node

[pfTree p250]

— **defun pfExport** —

```
(defun |pfExport| (pfitems)
 (|pfTree| '|Export| (list pfitems)))
```

———-

8.5.47 defun Construct an Expression leaf node

[pfLeaf p245]
[ifcar p**??**]

— **defun pfExpression** —

```
(defun |pfExpression| (expr &rest optpos)
 (|pfLeaf| '|expression| expr (ifcar optpos)))
```

———-

8.5.48 defun pfFirst

— **defun pfFirst 0** —

```
(defun |pfFirst| (form)
 (cadr form))
```

———-

8.5.49 defun Create an Application Fix node

[pfApplication p251]
[pfId p244]

— **defun pfFix** —

```
(defun |pfFix| (pf)
 (|pfApplication| (|pfId| 'Y) pf))
```

———-

8.5.50 defun Create a Free node

[pfTree p250]

— **defun pfFree** —

```
(defun |pfFree| (pfitems)
 (|pfTree| '|Free| (list pfitems)))
```

———-

8.5.51 defun Is this a Free node?

[pfAbSynOp? p410]

— **defun pfFree?** —

```
(defun |pfFree?| (pf)
 (|pfAbSynOp?| pf '|Free|))
```

———-

8.5.52 defun Return the parts of the Items of a Free node

[pfParts p247]
[pfFreeItems p264]

— **defun pf0FreeItems** —

```
(defun |pf0FreeItems| (pf)
 (|pfParts| (|pfFreeItems| pf)))
```

————-

8.5.53 defun Return the Items of a Free node

— **defun pfFreeItems 0** —

```
(defun |pfFreeItems| (pf)
 (cadr pf))
```

————-

8.5.54 defun Construct a Forin node

[pfTree p250]

— **defun pfForin** —

```
(defun |pfForin| (pflhs pfwhole)
 (|pfTree| '|Forin| (list pflhs pfwhole)))
```

————-

8.5.55 defun Is this a ForIn node?

[pfAbSynOp? p410]

— **defun pfForin?** —

```
(defun |pfForin?| (pf)
 (|pfAbSynOp?| pf '|Forin|))
```

————-

8.5.56 defun Return all the parts of the LHS of a ForIn node

[pfParts p247]
[pfForinLhs p265]

— **defun pf0ForinLhs** —

```
(defun |pf0ForinLhs| (pf)
 (|pfParts| (|pfForinLhs| pf)))
```

————-

8.5.57 defun Return the LHS part of a ForIn node

— **defun pfForinLhs 0** —

```
(defun |pfForinLhs| (pf)
 (cadr pf))
```

————-

8.5.58 defun Return the Whole part of a ForIn node

— **defun pfForinWhole 0** —

```
(defun |pfForinWhole| (pf)
 (caddr pf))
```

————-

8.5.59 defun pfFromDom

[pfApplication? p253]
[pfApplication p251]
[pfApplicationOp p252]
[pfApplicationArg p252]
[pfFromdom p266]

— **defun pfFromDom** —

```
(defun |pfFromDom| (dom expr)
 (cond
  ((|pfApplication?| expr)
   (|pfApplication|
    (|pfFromdom| (|pfApplicationOp| expr) dom)
    (|pfApplicationArg| expr)))
  (t (|pfFromdom| expr dom))))
```

———-

8.5.60 defun Construct a Fromdom node

[pfTree p250]

— **defun pfFromdom** —

```
(defun |pfFromdom| (pfwhat pfdomain)
 (|pfTree| '|Fromdom| (list pfwhat pfdomain)))
```

———-

8.5.61 defun Is this a Fromdom mode?

[pfAbSynOp? p410]

— **defun pfFromdom?** —

```
(defun |pfFromdom?| (pf)
 (|pfAbSynOp?| pf '|Fromdom|))
```

———-

8.5.62 defun Return the What part of a Fromdom node

— **defun pfFromdomWhat 0** —

```
(defun |pfFromdomWhat| (pf)
 (cadr pf))
```

———-

8.5.63 defun Return the Domain part of a Fromdom node

— **defun pfFromdomDomain 0** —

```
(defun |pfFromdomDomain| (pf)
 (caddr pf))
```

————-

8.5.64 defun Construct a Hide node

[pfTree p250]

— **defun pfHide** —

```
(defun |pfHide| (a part)
 (declare (ignore a))
 (|pfTree| '|Hide| (list part)))
```

————-

8.5.65 defun pfIf

[pfTree p250]

— **defun pfIf** —

```
(defun |pfIf| (pfcond pfthen pfelse)
 (|pfTree| '|If| (list pfcond pfthen pfelse)))
```

————-

8.5.66 defun Is this an If node?

[pfAbSynOp? p410]

— **defun pfIf?** —

```
(defun |pfIf?| (pf)
  (|pfAbSynOp?| pf '|If|))
```

————-

8.5.67 defun Return the Cond part of an If

— **defun pfIfCond 0** —

```
(defun |pfIfCond| (pf)
 (cadr pf))
```

————-

8.5.68 defun Return the Then part of an If

— **defun pfIfThen 0** —

```
(defun |pfIfThen| (pf)
 (caddr pf))
```

————-

8.5.69 defun pfIfThenOnly

[pfIf p267]
[pfNothing p243]

— **defun pfIfThenOnly** —

```
(defun |pfIfThenOnly| (pred cararg)
 (|pfIf| pred cararg (|pfNothing|)))
```

————-

8.5.70 defun Return the Else part of an If

— **defun pfIfElse 0** —

```
(defun |pfIfElse| (pf)
 (cadddr pf))
```

————-

8.5.71 defun Construct an Import node

[pfTree p250]

— **defun pfImport** —

```
(defun |pfImport| (pfitems)
 (|pfTree| '|Import| (list pfitems)))
```

———-

8.5.72 defun Construct an Iterate node

[pfTree p250]

— **defun pfIterate** —

```
(defun |pfIterate| (pffrom)
 (|pfTree| '|Iterate| (list pffrom)))
```

———-

8.5.73 defun Is this an Iterate node?

[pfAbSynOp? p410]

— **defun pfIterate?** —

```
(defun |pfIterate?| (pf)
 (|pfAbSynOp?| pf '|Iterate|))
```

———-

8.5.74 defun Handle an infix application

[pfListOf p243]
[pfIdSymbol p245]
[pfAnd p251]
[pfOr p278]
[pfApplication p251]
[pfTuple p290]

— **defun pfInfApplication** —

```
(defun |pfInfApplication| (op left right)
 (cond
  ((eq (|pfIdSymbol| op) '|and|) (|pfAnd| left right))
  ((eq (|pfIdSymbol| op) '|or|) (|pfOr| left right))
  (t (|pfApplication| op (|pfTuple| (|pfListOf| (list left right)))))))
```

———-

8.5.75 defun Create an Inline node

[pfTree p250]

— **defun pfInline** —

```
(defun |pfInline| (pfitems)
 (|pfTree| '|Inline| (list pfitems)))
```

———-

8.5.76 defun pfLam

[pfAbSynOp? p410]
[pfFirst p262]
[pfNothing p243]
[pfSecond p284]
[pfLambda p271]

— **defun pfLam** —

```
(defun |pfLam| (variable body)
 (let (bdy rets)
  (if (|pfAbSynOp?| body '|returntyped|)
   (setq rets (|pfFirst| body))
   (setq rets (|pfNothing|)))
  (if (|pfAbSynOp?| body '|returntyped|)
   (setq bdy (|pfSecond| body))
   (setq bdy body))
  (|pfLambda| variable rets bdy)))
```

———-

8.5.77 defun pfLambda

[pfTree p250]

— **defun pfLambda** —

```
(defun |pfLambda| (pfargs pfrets pfbody)
 (|pfTree| '|Lambda| (list pfargs pfrets pfbody)))
```

————-

8.5.78 defun Return the Body part of a Lambda node

— **defun pfLambdaBody 0** —

```
(defun |pfLambdaBody| (pf)
 (cadddr pf))
```

————-

8.5.79 defun Return the Rets part of a Lambda node

— **defun pfLambdaRets 0** —

```
(defun |pfLambdaRets| (pf)
 (caddr pf))
```

————-

8.5.80 defun Is this a Lambda node?

[pfAbSynOp? p410]

— **defun pfLambda?** —

```
(defun |pfLambda?| (pf)
 (|pfAbSynOp?| pf '|Lambda|))
```

————-

8.5.81 defun Return the Args part of a Lambda node

— **defun pfLambdaArgs 0** —

```
(defun |pfLambdaArgs| (pf)
 (cadr pf))
```

————-

8.5.82 defun Return the Args of a Lambda Node

[pfParts p247]
[pfLambdaArgs p272]

— **defun pf0LambdaArgs** —

```
(defun |pf0LambdaArgs| (pf)
 (|pfParts| (|pfLambdaArgs| pf)))
```

————-

8.5.83 defun Construct a Local node

[pfTree p250]

— **defun pfLocal** —

```
(defun |pfLocal| (pfitems)
 (|pfTree| '|Local| (list pfitems)))
```

————-

8.5.84 defun Is this a Local node?

[pfAbSynOp? p410]

— **defun pfLocal?** —

```
(defun |pfLocal?| (pf)
 (|pfAbSynOp?| pf '|Local|))
```

————-

8.5.85 defun Return the parts of Items of a Local node

[pfParts p247]
[pfLocalItems p273]

— **defun pf0LocalItems** —

```
(defun |pf0LocalItems| (pf)
 (|pfParts| (|pfLocalItems| pf)))
```

———-

8.5.86 defun Return the Items of a Local node

— **defun pfLocalItems 0** —

```
(defun |pfLocalItems| (pf)
 (cadr pf))
```

———-

8.5.87 defun Construct a Loop node

[pfTree p250]

— **defun pfLoop** —

```
(defun |pfLoop| (pfiterators)
 (|pfTree| '|Loop| (list pfiterators)))
```

———-

8.5.88 defun pfLoop1

[pfLoop p273]
[pfListOf p243]
[pfDo p260]

— **defun pfLoop1** —

```
(defun |pfLoop1| (body)
 (|pfLoop| (|pfListOf| (list (|pfDo| body)))))
```

———-

8.5.89 defun Is this a Loop node?

[pfAbSynOp? p410]

— **defun pfLoop?** —

```
(defun |pfLoop?| (pf)
 (|pfAbSynOp?| pf '|Loop|))
```

———-

8.5.90 defun Return the Iterators of a Loop node

— **defun pfLoopIterators 0** —

```
(defun |pfLoopIterators| (pf)
 (cadr pf))
```

———-

8.5.91 defun pf0LoopIterators

[pfParts p247]
[pf0LoopIterators p274]

— **defun pf0LoopIterators** —

```
(defun |pf0LoopIterators| (pf)
 (|pfParts| (|pfLoopIterators| pf)))
```

———-

8.5.92 defun pfLp

[pfLoop p273]
[pfListOf p243]
[pfDo p260]

— **defun pfLp** —

```
(defun |pfLp| (iterators body)
 (|pfLoop| (|pfListOf| (append iterators (list (|pfDo| body))))))
```

———-

8.5.93 defun Create a Macro node

[pfTree p250]

— **defun pfMacro** —

```
(defun |pfMacro| (pflhs pfrhs)
 (|pfTree| '|Macro| (list pflhs pfrhs)))
```

———-

8.5.94 defun Is this a Macro node?

[pfAbSynOp? p410]

— **defun pfMacro?** —

```
(defun |pfMacro?| (pf)
 (|pfAbSynOp?| pf '|Macro|))
```

———-

8.5.95 defun Return the Lhs of a Macro node

— **defun pfMacroLhs 0** —

```
(defun |pfMacroLhs| (pf)
 (cadr pf))
```

———-

8.5.96 defun Return the Rhs of a Macro node

— **defun pfMacroRhs 0** —

```
(defun |pfMacroRhs| (pf)
 (caddr pf))
```

———-

8.5.97 defun Construct an MLambda node

[pfTree p250]

— **defun pfMLambda** —

```
(defun |pfMLambda| (pfargs pfbody)
 (|pfTree| '|MLambda| (list pfargs pfbody)))
```

———-

8.5.98 defun Is this an MLambda node?

[pfAbSynOp? p410]

— **defun pfMLambda?** —

```
(defun |pfMLambda?| (pf)
 (|pfAbSynOp?| pf '|MLambda|))
```

———-

8.5.99 defun Return the Args of an MLambda

— **defun pfMLambdaArgs 0** —

```
(defun |pfMLambdaArgs| (pf)
 (cadr pf))
```

———-

8.5.100 defun Return the parts of an MLambda argument

[pfParts p247]

— **defun pf0MLambdaArgs** —

```
(defun |pf0MLambdaArgs| (pf)
 (|pfParts| (|pfMLambdaArgs| pf)))
```

———-

8.5.101 defun pfMLambdaBody

— **defun pfMLambdaBody 0** —

```
(defun |pfMLambdaBody| (pf)
 (caddr pf))
```

———-

8.5.102 defun Is this a Not node?

[pfAbSynOp? p410]

— **defun pfNot?** —

```
(defun |pfNot?| (pf)
 (|pfAbSynOp?| pf '|Not|))
```

———-

8.5.103 defun Return the Arg part of a Not node

— **defun pfNotArg 0** —

```
(defun |pfNotArg| (pf)
 (cadr pf))
```

———-

8.5.104 defun Construct a NoValue node

[pfTree p250]

— **defun pfNovalue** —

```
(defun |pfNovalue| (pfexpr)
 (|pfTree| '|Novalue| (list pfexpr)))
```

———-

8.5.105 defun Is this a Novalue node?

[pfAbSynOp? p410]

— **defun pfNovalue?** —

```
(defun |pfNovalue?| (pf)
 (|pfAbSynOp?| pf '|Novalue|))
```

———-

8.5.106 defun Return the Expr part of a Novalue node

— **defun pfNovalueExpr 0** —

```
(defun |pfNovalueExpr| (pf)
 (cadr pf))
```

———-

8.5.107 defun Construct an Or node

[pfTree p250]

— **defun pfOr** —

```
(defun |pfOr| (pfleft pfright)
 (|pfTree| '|Or| (list pfleft pfright)))
```

———-

8.5.108 defun Is this an Or node?

[pfAbSynOp? p410]

— **defun pfOr?** —

```
(defun |pfOr?| (pf)
 (|pfAbSynOp?| pf '|Or|))
```

———-

8.5.109 defun Return the Left part of an Or node

— **defun pfOrLeft 0** —

```
(defun |pfOrLeft| (pf)
 (cadr pf))
```

———-

8.5.110 defun Return the Right part of an Or node

— **defun pfOrRight 0** —

```
(defun |pfOrRight| (pf)
 (caddr pf))
```

———-

8.5.111 defun Return the part of a parenthesised expression

— **defun pfParen** —

```
(defun |pfParen| (a part)
 (declare (ignore a))
 part)
```

———-

8.5.112 defun pfPretend

[pfTree p250]

— **defun pfPretend** —

```
(defun |pfPretend| (pfexpr pftype)
 (|pfTree| '|Pretend| (list pfexpr pftype)))
```

———-

8.5.113 defun Is this a Pretend node?

[pfAbSynOp? p410]

— **defun pfPretend?** —

```
(defun |pfPretend?| (pf)
 (|pfAbSynOp?| pf '|Pretend|))
```

———-

8.5.114 defun Return the Expression part of a Pretend node

— **defun pfPretendExpr 0** —

```
(defun |pfPretendExpr| (pf)
 (cadr pf))
```

———-

8.5.115 defun Return the Type part of a Pretend node

— **defun pfPretendType 0** —

```
(defun |pfPretendType| (pf)
 (caddr pf))
```

———-

8.5.116 defun Construct a QualType node

[pfTree p250]

— **defun pfQualType** —

```
(defun |pfQualType| (pftype pfqual)
 (|pfTree| '|QualType| (list pftype pfqual)))
```

———-

8.5.117 defun Construct a Restrict node

[pfTree p250]

— **defun pfRestrict** —

```
(defun |pfRestrict| (pfexpr pftype)
 (|pfTree| '|Restrict| (list pfexpr pftype)))
```

———-

8.5.118 defun Is this a Restrict node?

[pfAbSynOp? p410]

— **defun pfRestrict?** —

```
(defun |pfRestrict?| (pf)
 (|pfAbSynOp?| pf '|Restrict|))
```

———-

8.5.119 defun Return the Expr part of a Restrict node

— **defun pfRestrictExpr 0** —

```
(defun |pfRestrictExpr| (pf)
 (cadr pf))
```

———-

8.5.120 defun Return the Type part of a Restrict node

— **defun pfRestrictType 0** —

```
(defun |pfRestrictType| (pf)
 (caddr pf))
```

———-

8.5.121 defun Construct a RetractTo node

[pfTree p250]

— **defun pfRetractTo** —

```
(defun |pfRetractTo| (pfexpr pftype)
 (|pfTree| '|RetractTo| (list pfexpr pftype)))
```

———-

8.5.122 defun Construct a Return node

[pfTree p250]

— **defun pfReturn** —

```
(defun |pfReturn| (pfexpr pffrom)
 (|pfTree| '|Return| (list pfexpr pffrom)))
```

———-

8.5.123 defun Is this a Return node?

[pfAbSynOp? p410]

— **defun pfReturn?** —

```
(defun |pfReturn?| (pf)
 (|pfAbSynOp?| pf '|Return|))
```

———-

8.5.124 defun Return the Expr part of a Return node

— **defun pfReturnExpr 0** —

```
(defun |pfReturnExpr| (pf)
 (cadr pf))
```

———-

8.5.125 defun pfReturnNoName

[pfReturn p282]
[pfNothing p243]

— **defun pfReturnNoName** —

```
(defun |pfReturnNoName| (|value|)
 (|pfReturn| |value| (|pfNothing|)))
```

———-

8.5.126 defun Construct a ReturnTyped node

[pfTree p250]

— **defun pfReturnTyped** —

```
(defun |pfReturnTyped| (type body)
 (|pfTree| '|returntyped| (list type body)))
```

———-

8.5.127 defun Construct a Rule node

[pfTree p250]

— **defun pfRule** —

```
(defun |pfRule| (pflhsitems pfrhs)
 (|pfTree| '|Rule| (list pflhsitems pfrhs)))
```

———-

8.5.128 defun Return the Lhs of a Rule node

— **defun pfRuleLhsItems 0** —

```
(defun |pfRuleLhsItems| (pf)
 (cadr pf))
```

————-

8.5.129 defun Return the Rhs of a Rule node

— **defun pfRuleRhs 0** —

```
(defun |pfRuleRhs| (pf)
 (caddr pf))
```

————-

8.5.130 defun Is this a Rule node?

[pfAbSynOp? p410]

— **defun pfRule?** —

```
(defun |pfRule?| (pf)
 (|pfAbSynOp?| pf '|Rule|))
```

————-

8.5.131 defun pfSecond

— **defun pfSecond 0** —

```
(defun |pfSecond| (form)
 (caddr form))
```

————-

8.5.132 defun Construct a Sequence node

[pfTree p250]

— **defun pfSequence** —

```
(defun |pfSequence| (pfargs)
 (|pfTree| '|Sequence| (list pfargs)))
```

————-

8.5.133 defun Return the Args of a Sequence node

— **defun pfSequenceArgs 0** —

```
(defun |pfSequenceArgs| (pf)
 (cadr pf))
```

————-

8.5.134 defun Is this a Sequence node?

[pfAbSynOp? p410]

— **defun pfSequence?** —

```
(defun |pfSequence?| (pf)
 (|pfAbSynOp?| pf '|Sequence|))
```

————-

8.5.135 defun Return the parts of the Args of a Sequence node

[pfParts p247]
[pfSequenceArgs p285]

— **defun pf0SequenceArgs** —

```
(defun |pf0SequenceArgs| (pf)
 (|pfParts| (|pfSequenceArgs| pf)))
```

————-

8.5.136 defun Create a Suchthat node

[pfTree p250]

— **defun pfSuchthat** —

```
(defun |pfSuchthat| (pfcond)
 (|pfTree| '|Suchthat| (list pfcond)))
```

———-

8.5.137 defun Is this a SuchThat node?

[pfAbSynOp? p410]

— **defun pfSuchthat?** —

```
(defun |pfSuchthat?| (pf)
 (|pfAbSynOp?| pf '|Suchthat|))
```

———-

8.5.138 defun Return the Cond part of a SuchThat node

— **defun pfSuchthatCond 0** —

```
(defun |pfSuchthatCond| (pf)
 (cadr pf))
```

———-

8.5.139 defun Create a Tagged node

[pfTree p250]

— **defun pfTagged** —

```
(defun |pfTagged| (pftag pfexpr)
 (|pfTree| '|Tagged| (list pftag pfexpr)))
```

———-

8.5.140 defun Is this a Tagged node?

[pfAbSynOp? p410]

— **defun pfTagged?** —

```
(defun |pfTagged?| (pf)
 (|pfAbSynOp?| pf '|Tagged|))
```

———-

8.5.141 defun Return the Expression portion of a Tagged node

— **defun pfTaggedExpr 0** —

```
(defun |pfTaggedExpr| (pf)
 (caddr pf))
```

———-

8.5.142 defun Return the Tag of a Tagged node

— **defun pfTaggedTag 0** —

```
(defun |pfTaggedTag| (pf)
 (cadr pf))
```

———-

8.5.143 defun pfTaggedToTyped

[pfTagged? p287]
[pfTaggedExpr p287]
[pfNothing p243]
[pfTaggedTag p287]
[pfId? p244]
[pfId p244]
[pfTyped p288]
[pfSuch p242]
[pfInfApplication p269]

— **defun pfTaggedToTyped** —

```
(defun |pfTaggedToTyped| (arg)
 (let (a form rt)
  (if (|pfTagged?| arg)
   (setq rt (|pfTaggedExpr| arg))
   (setq rt (|pfNothing|)))
  (if (|pfTagged?| arg)
   (setq form (|pfTaggedTag| arg))
   (setq form arg))
  (cond
   ((null (|pfId?| form))
    (setq a (|pfId| (gensym)))
    (|pfTyped| (|pfSuch| a (|pfInfApplication| (|pfId| '=) a form)) rt))
   (t (|pfTyped| form rt)))))
```

———-

8.5.144 defun pfTweakIf

[pfIfElse p268]
[pfNothing? p243]
[pfListOf p243]
[pfTree p250]
[pfIfCond p268]
[pfIfThen p268]

— **defun pfTweakIf** —

```
(defun |pfTweakIf| (form)
 (let (b a)
  (setq a (|pfIfElse| form))
  (setq b (if (|pfNothing?| a) (|pfListOf| NIL) a))
  (|pfTree| '|WIf| (list (|pfIfCond| form) (|pfIfThen| form) b))))
```

———-

8.5.145 defun Construct a Typed node

[pfTree p250]

— **defun pfTyped** —

```
(defun |pfTyped| (pfid pftype)
```

```
 (|pfTree| '|Typed| (list pfid pftype)))
```

———-

8.5.146 defun Is this a Typed node?

[pfAbSynOp? p410]

— **defun pfTyped?** —

```
(defun |pfTyped?| (pf)
 (|pfAbSynOp?| pf '|Typed|))
```

———-

8.5.147 defun Return the Type of a Typed node

— **defun pfTypedType 0** —

```
(defun |pfTypedType| (pf)
 (caddr pf))
```

———-

8.5.148 defun Return the Id of a Typed node

— **defun pfTypedId 0** —

```
(defun |pfTypedId| (pf)
 (cadr pf))
```

———-

8.5.149 defun Construct a Typing node

[pfTree p250]

— **defun pfTyping** —

```
(defun |pfTyping| (pfitems)
 (|pfTree| '|Typing| (list pfitems)))
```

———-

8.5.150 defun Return a Tuple node

[pfTree p250]

— **defun pfTuple** —

```
(defun |pfTuple| (pfparts)
 (|pfTree| '|Tuple| (list pfparts)))
```

———-

8.5.151 defun Return a Tuple from a List

[pfTuple p290]
[pfListOf p243]

— **defun pfTupleListOf** —

```
(defun |pfTupleListOf| (pfparts)
 (|pfTuple| (|pfListOf| pfparts)))
```

———-

8.5.152 defun Is this a Tuple node?

[pfAbSynOp? p410]

— **defun pfTuple?** —

```
(defun |pfTuple?| (pf)
 (|pfAbSynOp?| pf '|Tuple|))
```

———-

8.5.153 defun Return the Parts of a Tuple node

— **defun pfTupleParts 0** —

```
(defun |pfTupleParts| (pf)
 (cadr pf))
```

————-

8.5.154 defun Return the parts of a Tuple

[pfParts p247]
[pfTupleParts p291]

— **defun pf0TupleParts** —

```
(defun |pf0TupleParts| (pf)
 (|pfParts| (|pfTupleParts| pf)))
```

————-

8.5.155 defun Return a list from a Sequence node

[pfSequence? p285]
[pfAppend p253]
[pf0SequenceArgs p285]
[pfListOf p243]

— **defun pfUnSequence** —

```
(defun |pfUnSequence| (x)
 (if (|pfSequence?| x)
  (|pfListOf| (|pfAppend| (|pf0SequenceArgs| x)))
  (|pfListOf| x)))
```

————-

8.5.156 defun The comment is attached to all signatutres

[pfWDeclare p292]
[pfParts p247]

— **defun pfWDec** —

```
(defun |pfWDec| (doc name)
 (mapcar #'(lambda (i) (|pfWDeclare| i doc)) (|pfParts| name)))
```

———-

8.5.157 defun Construct a WDeclare node

[pfTree p250]

— **defun pfWDeclare** —

```
(defun |pfWDeclare| (pfsignature pfdoc)
 (|pfTree| '|WDeclare| (list pfsignature pfdoc)))
```

———-

8.5.158 defun Construct a Where node

[pfTree p250]

— **defun pfWhere** —

```
(defun |pfWhere| (pfcontext pfexpr)
 (|pfTree| '|Where| (list pfcontext pfexpr)))
```

———-

8.5.159 defun Is this a Where node?

[pfAbSynOp? p410]

— **defun pfWhere?** —

```
(defun |pfWhere?| (pf)
 (|pfAbSynOp?| pf '|Where|))
```

———-

8.5.160 defun Return the parts of the Context of a Where node

[pfParts p247]
[pfWhereContext p293]

— **defun pf0WhereContext** —

```
(defun |pf0WhereContext| (pf)
 (|pfParts| (|pfWhereContext| pf)))
```

————-

8.5.161 defun Return the Context of a Where node

— **defun pfWhereContext 0** —

```
(defun |pfWhereContext| (pf)
 (cadr pf))
```

————-

8.5.162 defun Return the Expr part of a Where node

— **defun pfWhereExpr 0** —

```
(defun |pfWhereExpr| (pf)
 (caddr pf))
```

————-

8.5.163 defun Construct a While node

[pfTree p250]

— **defun pfWhile** —

```
(defun |pfWhile| (pfcond)
 (|pfTree| '|While| (list pfcond)))
```

————-

8.5.164 defun Is this a While node?

[pfAbSynOp? p410]

— **defun pfWhile?** —

```
(defun |pfWhile?| (pf)
 (|pfAbSynOp?| pf '|While|))
```

———-

8.5.165 defun Return the Cond part of a While node

— **defun pfWhileCond 0** —

```
(defun |pfWhileCond| (pf)
 (cadr pf))
```

———-

8.5.166 defun Construct a With node

[pfTree p250]

— **defun pfWith** —

```
(defun |pfWith| (pfbase pfwithin pfwithon)
 (|pfTree| '|With| (list pfbase pfwithin pfwithon)))
```

———-

8.5.167 defun Create a Wrong node

[pfTree p250]

— **defun pfWrong** —

```
(defun |pfWrong| (pfwhy pfrubble)
 (|pfTree| '|Wrong| (list pfwhy pfrubble)))
```

———-

8.5.168 defun Is this a Wrong node?

[pfAbSynOp? p410]

— **defun pfWrong?** —

```
(defun |pfWrong?| (pf)
 (|pfAbSynOp?| pf '|Wrong|))
```

————-

Chapter 9

Pftree to s-expression translation

Pftree to s-expression translation. Used to interface the new parser technology to the interpreter. The input is a parseTree and the output is an old-parser-style s-expression.

9.0.169 defun Pftree to s-expression translation

[pf2Sex1 p298]
[$insideSEQ p??]
[$insideApplication p??]
[$insideRule p??]
[$QuietCommand p47]

— **defun pf2Sex** —

```
(defun |pf2Sex| (pf)
 (let (|$insideSEQ| |$insideApplication| |$insideRule|)
 (declare (special |$insideSEQ| |$insideApplication| |$insideRule|
                   |$QuietCommand|))
  (setq |$QuietCommand| nil)
  (setq |$insideRule| nil)
  (setq |$insideApplication| nil)
  (setq |$insideSEQ| nil)
  (|pf2Sex1| pf)))
```

———-

9.0.170 defun Pftree to s-expression translation inner function

[pfNothing? p243]
[pfSymbol? p250]
[pfSymbolSymbol p250]
[pfLiteral? p246]
[pfLiteral2Sex p302]
[pfIdSymbol p245]
[pfApplication? p253]
[pfApplication2Sex p303]
[pfTuple? p290]
[pf2Sex1 p298]
[pf0TupleParts p291]
[pfIf? p267]
[pfIfCond p268]
[pfIfThen p268]
[pfIfElse p268]
[pfTagged? p287]
[pfTaggedTag p287]
[pfTaggedExpr p287]
[pfCoerceto? p257]
[pfCoercetoExpr p257]
[pfCoercetoType p257]
[pfPretend? p280]
[pfPretendExpr p280]
[pfPretendType p280]
[pfFromdom? p266]
[opTran p321]
[pfFromdomWhat p266]
[pfFromdomDomain p267]
[pfSequence? p285]
[pfSequence2Sex p308]
[pfExit? p261]
[pfExitCond p261]
[pfExitExpr p262]
[pfLoop? p274]
[loopIters2Sex p309]
[pf0LoopIterators p274]
[pfCollect? p258]
[pfCollect2Sex p312]
[pfForin? p264]
[pf0ForinLhs p265]
[pfForinWhole p265]
[pfWhile? p294]
[pfWhileCond p294]
[pfSuchthat? p286]

[keyedSystemError p**??**]
[pfSuchthatCond p286]
[pfDo? p260]
[pfDoBody p260]
[pfTyped? p289]
[pfTypedType p289]
[pfTypedId p289]
[pfAssign? p254]
[pf0AssignLhsItems p254]
[pfAssignRhs p254]
[pfDefinition? p259]
[pfDefinition2Sex p313]
[pfLambda? p271]
[pfLambda2Sex p315]
[pfMLambda? p276]
[pfRestrict? p281]
[pfRestrictExpr p281]
[pfRestrictType p281]
[pfFree? p263]
[pf0FreeItems p264]
[pfLocal? p272]
[pf0LocalItems p273]
[pfWrong? p295]
[spadThrow p**??**]
[pfAnd? p252]
[pfAndLeft p252]
[pfAndRight p253]
[pfOr? p278]
[pfOrLeft p279]
[pfOrRight p279]
[pfNot? p277]
[pfNotArg p277]
[pfNovalue? p278]
[pfNovalueExpr p278]
[pfRule? p284]
[pfRule2Sex p316]
[pfBreak? p256]
[pfBreakFrom p256]
[pfMacro? p275]
[pfReturn? p282]
[pfReturnExpr p282]
[pfIterate? p269]
[pfWhere? p292]
[pf0WhereContext p293]
[pfWhereExpr p293]
[pfAbSynOp p410]

[tokPart p411]
[$insideSEQ p??]
[$insideRule p??]
[$QuietCommand p47]

— defun pf2Sex1 —

```
(defun |pf2Sex1| (pf)
 (let (args idList type op tagPart tag s)
 (declare (special |$insideSEQ| |$insideRule| |$QuietCommand|))
  (cond
   ((|pfNothing?| pf) '|noBranch|)
   ((|pfSymbol?| pf)
    (if (eq |$insideRule| '|left|)
     (progn
      (setq s (|pfSymbolSymbol| pf))
      (list '|constant| (list 'quote s)))
     (list 'quote (|pfSymbolSymbol| pf))))
   ((|pfLiteral?| pf) (|pfLiteral2Sex| pf))
   ((|pfId?| pf)
     (if |$insideRule|
      (progn
        (setq s (|pfIdSymbol| pf))
        (if (member s '(|%pi| |%e| |%i|))
          s
          (list 'quote s)))
      (|pfIdSymbol| pf)))
   ((|pfApplication?| pf) (|pfApplication2Sex| pf))
   ((|pfTuple?| pf) (cons '|Tuple| (mapcar #'|pf2Sex1| (|pf0TupleParts| pf))))
   ((|pfIf?| pf)
     (list 'if (|pf2Sex1| (|pfIfCond| pf))
               (|pf2Sex1| (|pfIfThen| pf))
               (|pf2Sex1| (|pfIfElse| pf))))
   ((|pfTagged?| pf)
    (setq tag (|pfTaggedTag| pf))
    (setq tagPart
     (if (|pfTuple?| tag)
      (cons '|Tuple| (mapcar #'|pf2Sex1| (|pf0TupleParts| tag)))
      (|pf2Sex1| tag)))
    (list '|:| tagPart (|pf2Sex1| (|pfTaggedExpr| pf))))
   ((|pfCoerceto?| pf)
    (list '|::| (|pf2Sex1| (|pfCoercetoExpr| pf))
                (|pf2Sex1| (|pfCoercetoType| pf))))
   ((|pfPretend?| pf)
    (list '|pretend| (|pf2Sex1| (|pfPretendExpr| pf))
                     (|pf2Sex1| (|pfPretendType| pf))))
   ((|pfFromdom?| pf)
    (setq op (|opTran| (|pf2Sex1| (|pfFromdomWhat| pf))))
    (when (eq op '|braceFromCurly|) (setq op 'seq))
```

```
 (list '|$elt| (|pf2Sex1| (|pfFromdomDomain| pf)) op))
((|pfSequence?| pf) (|pfSequence2Sex| pf))
((|pfExit?| pf)
 (if |$insideSEQ|
  (list '|exit| (|pf2Sex1| (|pfExitCond| pf))
                (|pf2Sex1| (|pfExitExpr| pf)))
  (list 'if (|pf2Sex1| (|pfExitCond| pf))
            (|pf2Sex1| (|pfExitExpr| pf)) '|noBranch|)))
((|pfLoop?| pf) (cons 'repeat (|loopIters2Sex| (|pf0LoopIterators| pf))))
((|pfCollect?| pf) (|pfCollect2Sex| pf))
((|pfForin?| pf)
 (cons 'in
  (append (mapcar #'|pf2Sex1| (|pf0ForinLhs| pf))
          (list (|pf2Sex1| (|pfForinWhole| pf))))))
((|pfWhile?| pf) (list 'while (|pf2Sex1| (|pfWhileCond| pf))))
((|pfSuchthat?| pf)
 (if (eq |$insideRule| '|left|)
  (|keyedSystemError| "S2GE0017" (list "pf2Sex1: pfSuchThat"))
  (list '|\|| (|pf2Sex1| (|pfSuchthatCond| pf)))))
((|pfDo?| pf) (|pf2Sex1| (|pfDoBody| pf)))
((|pfTyped?| pf)
 (setq type (|pfTypedType| pf))
 (if (|pfNothing?| type)
  (|pf2Sex1| (|pfTypedId| pf))
  (list '|:| (|pf2Sex1| (|pfTypedId| pf)) (|pf2Sex1| (|pfTypedType| pf)))))
((|pfAssign?| pf)
 (setq idList (mapcar #'|pf2Sex1| (|pf0AssignLhsItems| pf)))
 (if (not (eql (length idList) 1))
  (setq idList (cons '|Tuple| idList))
  (setq idList (car idList)))
 (list 'let idList (|pf2Sex1| (|pfAssignRhs| pf))))
((|pfDefinition?| pf) (|pfDefinition2Sex| pf))
((|pfLambda?| pf) (|pfLambda2Sex| pf))
((|pfMLambda?| pf) '|/throwAway|)
((|pfRestrict?| pf)
 (list '@ (|pf2Sex1| (|pfRestrictExpr| pf))
          (|pf2Sex1| (|pfRestrictType| pf))))
((|pfFree?| pf) (cons '|free| (mapcar #'|pf2Sex1| (|pf0FreeItems| pf))))
((|pfLocal?| pf) (cons '|local| (mapcar #'|pf2Sex1| (|pf0LocalItems| pf))))
((|pfWrong?| pf) (|spadThrow|))
((|pfAnd?| pf)
 (list '|and| (|pf2Sex1| (|pfAndLeft| pf))
              (|pf2Sex1| (|pfAndRight| pf))))
((|pfOr?| pf)
 (list '|or| (|pf2Sex1| (|pfOrLeft| pf))
             (|pf2Sex1| (|pfOrRight| pf))))
((|pfNot?| pf) (list '|not| (|pf2Sex1| (|pfNotArg| pf))))
((|pfNovalue?| pf)
 (setq |$QuietCommand| t)
 (list 'seq (|pf2Sex1| (|pfNovalueExpr| pf))))
```

```
   ((|pfRule?| pf) (|pfRule2Sex| pf))
   ((|pfBreak?| pf) (list '|break| (|pfBreakFrom| pf)))
   ((|pfMacro?| pf) '|/throwAway|)
   ((|pfReturn?| pf) (list '|return| (|pf2Sex1| (|pfReturnExpr| pf))))
   ((|pfIterate?| pf) (list '|iterate|))
   ((|pfWhere?| pf)
    (setq args (mapcar #'|pf2Sex1| (|pf0WhereContext| pf)))
    (if (eql (length args) 1)
     (cons '|where| (cons (|pf2Sex1| (|pfWhereExpr| pf)) args))
     (list '|where| (|pf2Sex1| (|pfWhereExpr| pf)) (cons 'seq args))))
;  -- under strange circumstances/piling, system commands can wind
;  -- up in expressions. This just passes it through as a string for
;  -- the user to figure out what happened.
   ((eq (|pfAbSynOp| pf) '|command|) (|tokPart| pf))
   (t (|keyedSystemError| "S2GE0017" (list "pf2Sex1"))))))
```

———-

9.0.171 defun Convert a Literal to an S-expression

[pfLiteralClass p246]
[pfLiteralString p247]
[float2Sex p303]
[pfSymbolSymbol p250]
[pfLeafToken p246]
[keyedSystemError p**??**]
[$insideRule p**??**]

— defun pfLiteral2Sex —

```
(defun |pfLiteral2Sex| (pf)
 (let (s type)
 (declare (special |$insideRule|))
  (setq type (|pfLiteralClass| pf))
  (cond
   ((eq type '|integer|) (read-from-string (|pfLiteralString| pf)))
   ((or (eq type '|string|) (eq type '|char|))
    (|pfLiteralString| pf))
   ((eq type '|float|) (|float2Sex| (|pfLiteralString| pf)))
   ((eq type '|symbol|)
    (if |$insideRule|
     (progn
      (setq s (|pfSymbolSymbol| pf))
      (list 'quote s))
     (|pfSymbolSymbol| pf)))
   ((eq type '|expression|) (list 'quote (|pfLeafToken| pf)))
   (t
```

```
    (|keyedSystemError| 'S2GE0017 (list "pfLiteral2Sex: unexpected form"))))))
```

———-

9.0.172 defun Convert a float to an S-expression

[$useBFasDefault p??]

— **defun float2Sex** —

```
(defun |float2Sex| (num)
 (let (exp frac bfForm fracPartString intPart dotIndex expPart mantPart eIndex)
 (declare (special |$useBFasDefault|))
  (setq eIndex (search "e" num))
  (if eIndex
   (setq mantPart (subseq num 0 eIndex))
   (setq mantPart num))
  (if eIndex
   (setq expPart (read-from-string (subseq num (+ eIndex 1))))
   (setq expPart 0))
  (setq dotIndex (search "." mantPart))
  (if dotIndex
   (setq intPart (read-from-string (subseq mantPart 0 dotIndex)))
   (setq intPart (read-from-string mantPart)))
  (if dotIndex
   (setq fracPartString (subseq mantPart (+ dotIndex 1)))
   (setq fracPartString 0))
  (setq bfForm
   (make-float intPart (read-from-string fracPartString)
                       (length fracPartString) expPart))
  (if |$useBFasDefault|
   (progn
    (setq frac (cadr bfForm))
    (setq exp (cddr bfForm))
    (list (list '|$elt| (list '|Float|) '|float|) frac exp 10))
   bfForm)))
```

———-

9.0.173 defun Change an Application node to an S-expression

[pfOp2Sex p306]
[pfApplicationOp p252]
[opTran p321]
[pf0TupleParts p291]

[pfApplicationArg p252]
[pfTuple? p290]
[pf2Sex1 p298]
[pf2Sex p297]
[pfSuchThat2Sex p305]
[hasOptArgs? p307]
[$insideApplication p??]
[$insideRule p??]

— defun pfApplication2Sex —

```
(defun |pfApplication2Sex| (pf)
 (let (|$insideApplication| x val realOp tmp1 qt argSex typeList args op)
 (declare (special |$insideApplication| |$insideRule|))
  (setq |$insideApplication| t)
  (setq op (|pfOp2Sex| (|pfApplicationOp| pf)))
  (setq op (|opTran| op))
  (cond
   ((eq op '->)
     (setq args (|pf0TupleParts| (|pfApplicationArg| pf)))
     (if (|pfTuple?| (car args))
      (setq typeList (mapcar #'|pf2Sex1| (|pf0TupleParts| (car args))))
      (setq typeList (list (|pf2Sex1| (car args)))))
     (setq args (cons (|pf2Sex1| (cadr args)) typeList))
     (cons '|Mapping| args))
   ((and (eq op '|:|) (eq |$insideRule| '|left|))
    (list '|multiple| (|pf2Sex| (|pfApplicationArg| pf))))
   ((and (eq op '?) (eq |$insideRule| '|left|))
    (list '|optional| (|pf2Sex| (|pfApplicationArg| pf))))
   (t
    (setq args (|pfApplicationArg| pf))
    (cond
     ((|pfTuple?| args)
      (if (and (eq op '|\||) (eq |$insideRule| '|left|))
       (|pfSuchThat2Sex| args)
       (progn
        (setq argSex (cdr (|pf2Sex1| args)))
        (cond
         ((eq op '>) (list '< (cadr argSex) (car argSex)))
         ((eq op '>=) (list '|not| (list '< (car argSex) (cadr argSex))))
         ((eq op '<=) (list '|not| (list '< (cadr argSex) (car argSex))))
         ((eq op 'and) (list '|and| (car argSex) (cadr argSex)))
         ((eq op 'or) (list '|or| (car argSex) (cadr argSex)))
         ((eq op '|Iterate|) (list '|iterate|))
         ((eq op '|by|) (cons 'by argSex))
         ((eq op '|braceFromCurly|)
          (if (and (consp argSex) (eq (car argSex) 'seq))
           argSex
           (cons 'seq argSex)))
```

```
        ((and (consp op)
              (progn
               (setq qt (car op))
               (setq tmp1 (cdr op))
               (and (consp tmp1)
                    (eq (cdr tmp1) nil)
                    (progn
                     (setq realOp (car tmp1))
                     t)))
              (eq qt 'quote))
         (cons '|applyQuote| (cons op argSex)))
        ((setq val (|hasOptArgs?| argSex)) (cons op val))
        (t (cons op argSex))))))
    ((and (consp op)
          (progn
           (setq qt (car op))
           (setq tmp1 (cdr op))
           (and (consp tmp1)
                (eq (cdr tmp1) NIL)
                (progn
                 (setq realOp (car tmp1))
                 t)))
          (eq qt 'quote))
     (list '|applyQuote| op (|pf2Sex1| args)))
    ((eq op '|braceFromCurly|)
     (setq x (|pf2Sex1| args))
     (if (and (consp x) (eq (car x) 'seq))
      x
      (list 'seq x)))
    ((eq op '|by|) (list 'by (|pf2Sex1| args)))
    (t (list op (|pf2Sex1| args))))))))
```

———-

9.0.174 defun Convert a SuchThat node to an S-expression

[pf0TupleParts p291]
[pf2Sex1 p298]
[pf2Sex p297]
[$predicateList p??]

— **defun pfSuchThat2Sex** —

```
(defun |pfSuchThat2Sex| (args)
 (let (rhsSex lhsSex argList name)
 (declare (special |$predicateList|))
  (setq name (gentemp))
```

```
  (setq argList (|pf0TupleParts| args))
  (setq lhsSex (|pf2Sex1| (car argList)))
  (setq rhsSex (|pf2Sex| (cadr argList)))
  (setq |$predicateList|
   (cons (cons name (cons lhsSex rhsSex)) |$predicateList|))
  name))
```

———-

9.0.175 defun pfOp2Sex

[pf2Sex1 p298]
[pmDontQuote? p307]
[pfSymbol? p250]
[$quotedOpList p??]
[$insideRule p??]

— defun pfOp2Sex —

```
(defun |pfOp2Sex| (pf)
 (let (realOp tmp1 op alreadyQuoted)
 (declare (special |$quotedOpList| |$insideRule|))
  (setq alreadyQuoted (|pfSymbol?| pf))
  (setq op (|pf2Sex1| pf))
  (cond
   ((and (consp op)
         (eq (car op) 'quote)
         (progn
          (setq tmp1 (cdr op))
          (and (consp tmp1)
               (eq (cdr tmp1) nil)
               (progn
                (setq realOp (car tmp1)) t))))
    (cond
     ((eq |$insideRule| '|left|) realOp)
     ((eq |$insideRule| '|right|)
      (cond
       ((|pmDontQuote?| realOp) realOp)
       (t
        (setq |$quotedOpList| (cons op |$quotedOpList|))
        op)))
     ((eq realOp '|\||) realOp)
     ((eq realOp '|:|) realOp)
     ((eq realOp '?) realOp)
     (t op)))
   (t op))))
```

———-

9.0.176 defun pmDontQuote?

— defun pmDontQuote? 0 —

```
(defun |pmDontQuote?| (sy)
 (member sy
  '(+ - * ** ^ / |log| |exp| |pi| |sqrt| |ei| |li| |erf| |ci|
      |si| |dilog| |sin| |cos| |tan| |cot| |sec| |csc| |asin|
      |acos| |atan| |acot| |asec| |acsc| |sinh| |cosh| |tanh|
      |coth| |sech| |csch| |asinh| |acosh| |atanh| |acoth|
      |asech| |acsc|)))
```

———-

9.0.177 defun hasOptArgs?

— defun hasOptArgs? 0 —

```
(defun |hasOptArgs?| (argSex)
 (let (rhs  lhs opt nonOpt tmp1 tmp2)
 (dolist (arg argSex)
  (cond
   ((and (consp arg)
         (eq (car arg) 'optarg)
         (progn
          (setq tmp1 (cdr arg))
          (and (consp tmp1)
               (progn
                (setq lhs (car tmp1))
                (setq tmp2 (cdr tmp1))
                (and (consp tmp2)
                     (eq (cdr tmp2) nil)
                     (progn
                      (setq rhs (car tmp2))
                      t))))))
    (setq opt (cons (list lhs rhs) opt)))
   (t (setq nonOpt (cons arg nonOpt)))))
 (when opt
  (nconc (nreverse nonOpt) (list (cons '|construct| (nreverse opt)))))))
```

———-

9.0.178 defun Convert a Sequence node to an S-expression

[pf2Sex1 p298]
[pf0SequenceArgs p285]
[$insideSEQ p??]

— **defun pfSequence2Sex** —

```
(defun |pfSequence2Sex| (pf)
 (let (|$insideSEQ| tmp1 ruleList seq)
 (declare (special |$insideSEQ|))
  (setq |$insideSEQ| t)
  (setq seq (|pfSequence2Sex0| (mapcar #'|pf2Sex1| (|pf0SequenceArgs| pf))))
  (cond
   ((and (consp seq)
         (eq (car seq) 'seq)
         (progn (setq ruleList (cdr seq)) 't)
         (consp ruleList)
         (progn
          (setq tmp1 (car ruleList))
          (and (consp tmp1) (eq (car tmp1) '|rule|))))
    (list '|ruleset| (cons '|construct| ruleList)))
   (t seq))))
```

———-

9.0.179 defun pfSequence2Sex0

TPDHERE: rewrite this using (dolist (item seqList)...)

```
;pfSequence2Sex0 seqList ==
;  null seqList => "noBranch"
;  seqTranList := []
;  while seqList ^= nil repeat
;    item := first seqList
;    item is ["exit", cond, value] =>
;      item := ["IF", cond, value, pfSequence2Sex0 rest seqList]
;      seqTranList := [item, :seqTranList]
;      seqList := nil
;    seqTranList := [item ,:seqTranList]
;    seqList := rest seqList
;  #seqTranList = 1 => first seqTranList
;  ["SEQ", :nreverse seqTranList]
```

[pfSequence2Sex0 p308]

— **defun pfSequence2Sex0** —

```
(defun |pfSequence2Sex0| (seqList)
 (let (value tmp2 cond tmp1 item seqTranList)
  (if (null seqList)
   '|noBranch|
    (progn
     ((lambda ()
       (loop
        (if (not seqList)
         (return nil)
         (progn
           (setq item (car seqList))
           (cond
            ((and (consp item)
                  (eq (car item) '|exit|)
                  (progn
                   (setq tmp1 (cdr item))
                   (and (consp tmp1)
                        (progn
                         (setq cond (car tmp1))
                         (setq tmp2 (cdr tmp1))
                         (and (consp tmp2)
                              (eq (cdr tmp2) nil)
                              (progn
                               (setq value (car tmp2))
                               t))))))
               (setq item
                (list 'if cond value (|pfSequence2Sex0| (cdr seqList))))
               (setq seqTranList (cons item seqTranList))
               (setq seqList nil))
            (t
             (progn
              (setq seqTranList (cons item seqTranList))
              (setq seqList (cdr seqList))))))))))
     (if (eql (length seqTranList) 1)
      (car seqTranList)
      (cons 'seq (nreverse seqTranList)))))))
```

———-

9.0.180 defun Convert a loop node to an S-expression

TPDHERE: rewrite using dsetq

```
;loopIters2Sex iterList ==
;  result := nil
;  for iter in iterList repeat
;    sex := pf2Sex1 iter
;    sex is ['IN, var, ['SEGMENT, i, ["BY", incr]]] =>
```

```
;      result := [ ['STEP, var, i, incr], :result]
;    sex is ['IN, var, ["BY", ['SEGMENT, i, j], incr]] =>
;      result := [ ['STEP, var, i, incr, j], :result]
;    sex is ['IN, var, ['SEGMENT, i, j]] =>
;      result := [ ['STEP, var, i, 1, j], :result]
;    result := [sex, :result]
;  nreverse result
```

[pf2Sex1 p298]

— defun loopIters2Sex —

```
(defun |loopIters2Sex| (iterList)
 (let (j incr i var sex result tmp1 tmp2 tmp3 tmp4 tmp5 tmp6 tmp7 tmp8)
  (dolist (iter iterList (nreverse result))
   (setq sex (|pf2Sex1| iter))
   (cond
    ((and (consp sex)
          (eq (car sex) 'in)
          (progn
           (setq tmp1 (cdr sex))
           (and (consp tmp1)
                (progn
                 (setq var (car tmp1))
                 (setq tmp2 (cdr tmp1))
                 (and (consp tmp2)
                      (eq (cdr tmp2) nil)
                      (progn
                       (setq tmp3 (car tmp2))
                       (and (consp tmp3)
                            (eq (car tmp3) 'segment)
                            (progn
                             (setq tmp4 (cdr tmp3))
                             (and (consp tmp4)
                                  (progn
                                   (setq i (car tmp4))
                                   (setq tmp5 (cdr tmp4))
                                   (and (consp tmp5)
                                        (eq (cdr tmp5) nil)
                                        (progn
                                         (setq tmp6 (car tmp5))
                                         (and (consp tmp6)
                                              (eq (car tmp6) 'by)
                                              (progn
                                               (setq tmp7 (cdr tmp6))
                                               (and (consp tmp7)
                                                    (eq (cdr tmp7) nil)
                                                    (progn
                                                     (setq incr (car tmp7))
                                                     t)))))))))))))))))
```

```
 (setq result (cons (list 'step var i incr) result)))
((and (consp sex)
      (eq (car sex) 'in)
      (progn
       (setq tmp1 (cdr sex))
       (and (consp tmp1)
            (progn
             (setq var (car tmp1))
             (setq tmp2 (cdr tmp1))
             (and (consp tmp2)
                  (eq (cdr tmp2) nil)
                  (progn
                   (setq tmp3 (car tmp2))
                   (and (consp tmp3)
                        (eq (car tmp3) 'by)
                        (progn
                         (setq tmp4 (cdr tmp3))
                         (and (consp tmp4)
                          (progn
                           (setq tmp5 (car tmp4))
                           (and (consp tmp5)
                                (eq (car tmp5) 'segment)
                                (progn
                                 (setq tmp6 (cdr tmp5))
                                 (and (consp tmp6)
                                      (progn
                                       (setq i (car tmp6))
                                       (setq tmp7 (cdr tmp6))
                                       (and (consp tmp7)
                                            (eq (cdr tmp7) nil)
                                            (progn
                                             (setq j (car tmp7))
                                             t)))))))
                          (progn
                           (setq tmp8 (cdr tmp4))
                           (and (consp tmp8)
                                (eq (cdr tmp8) nil)
                                (progn
                                 (setq incr (car tmp8))
                                 t)))))))))))))
 (setq result (cons (list 'step var i incr j) result)))
((and (consp sex)
      (eq (car sex) 'in)
      (progn
       (setq tmp1 (cdr sex))
       (and (consp tmp1)
            (progn
             (setq var (car tmp1))
             (setq tmp2 (cdr tmp1))
             (and (consp tmp2)
```

```
                    (eq (cdr tmp2) nil)
                    (progn
                     (setq tmp3 (car tmp2))
                     (and (consp tmp3)
                          (eq (car tmp3) 'segment)
                          (progn
                           (setq tmp4 (cdr tmp3))
                           (and (consp tmp4)
                                (progn
                                 (setq i (car tmp4))
                                 (setq tmp5 (cdr tmp4))
                                 (and (consp tmp5)
                                      (eq (cdr tmp5) nil)
                                      (progn
                                       (setq j (car tmp5))
                                       t)))))))))))
      (setq result (cons (list 'step var i 1 j) result)))
     (t (setq result (cons sex result)))))))
```

———-

9.0.181 defun Change a Collect node to an S-expression

[loopIters2Sex p309]
[pfParts p247]
[pfCollectIterators p258]
[pf2Sex1 p298]
[pfCollectBody p258]

— **defun pfCollect2Sex** —

```
(defun |pfCollect2Sex| (pf)
 (let (var cond sex tmp1 tmp2 tmp3 tmp4)
  (setq sex
   (cons 'collect
    (append (|loopIters2Sex| (|pfParts| (|pfCollectIterators| pf)))
            (list (|pf2Sex1| (|pfCollectBody| pf))))))
  (cond
   ((and (consp sex)
         (eq (car sex) 'collect)
         (progn
          (setq tmp1 (cdr sex))
          (and (consp tmp1)
               (progn
                (setq tmp2 (car tmp1))
                (and (consp tmp2)
                     (eq (car tmp2) '|\||)
```

```
                        (progn
                         (setq tmp3 (cdr tmp2))
                         (and (consp tmp3)
                              (eq (cdr tmp3) nil)
                              (progn
                               (setq cond (car tmp3))
                               t)))))
                (progn
                 (setq tmp4 (cdr tmp1))
                 (and (consp tmp4)
                      (eq (cdr tmp4) nil)
                          (progn (setq var (car tmp4)) t)))))
          (symbolp var))
     (list '|\|| var cond))
    (t sex))))
```

———-

9.0.182 defun Convert a Definition node to an S-expression

[pf2Sex1 p298]
[pf0DefinitionLhsItems p260]
[pfDefinitionRhs p259]
[systemError p**??**]
[pfLambdaTran p314]
[$insideApplication p**??**]

— **defun pfDefinition2Sex** —

```
(defun |pfDefinition2Sex| (pf)
 (let (body argList tmp1 rhs id idList)
 (declare (special |$insideApplication|))
  (if |$insideApplication|
   (list 'optarg
    (|pf2Sex1| (car (|pf0DefinitionLhsItems| pf)))
    (|pf2Sex1| (|pfDefinitionRhs| pf)))
   (progn
    (setq idList (mapcar #'|pf2Sex1| (|pf0DefinitionLhsItems| pf)))
    (if (not (eql (length idList) 1))
     (|systemError|
       "lhs of definition must be a single item in the interpreter")
     (progn
      (setq id (car idList))
      (setq rhs (|pfDefinitionRhs| pf))
      (setq tmp1 (|pfLambdaTran| rhs))
      (setq argList (car tmp1))
      (setq body (cdr tmp1))
```

```
      (cons 'def
       (cons
        (if (eq argList '|id|)
         id
         (cons id argList))
        body))))))))
```

———-

9.0.183 defun Convert a Lambda node to an S-expression

[pfLambda? p271]
[pf0LambdaArgs p272]
[pfTyped? p289]
[pfCollectArgTran p315]
[pfTypedId p289]
[pfNothing? p243]
[pfTypedType p289]
[pf2Sex1 p298]
[systemError p??]
[pfLambdaRets p271]
[pfLambdaBody p271]

— **defun pfLambdaTran** —

```
(defun |pfLambdaTran| (pf)
 (let (retType argList argTypeList)
  (cond
   ((|pfLambda?| pf)
    (dolist (arg (|pf0LambdaArgs| pf))
     (if (|pfTyped?| arg)
      (progn
       (setq argList
        (cons (|pfCollectArgTran| (|pfTypedId| arg)) argList))
       (if (|pfNothing?| (|pfTypedType| arg))
        (setq argTypeList (cons nil argTypeList))
        (setq argTypeList
         (cons (|pf2Sex1| (|pfTypedType| arg)) argTypeList))))
      (|systemError| "definition args should be typed")))
    (setq argList (nreverse argList))
    (unless (|pfNothing?| (|pfLambdaRets| pf))
     (setq retType (|pf2Sex1| (|pfLambdaRets| pf))))
    (setq argTypeList (cons retType (nreverse argTypeList)))
    (cons argList
     (list argTypeList
           (mapcar #'(lambda (x) (declare (ignore x)) nil) argTypeList)
           (|pf2Sex1| (|pfLambdaBody| pf)))))
```

```
   (t (cons '|id| (list '(nil) '(nil) (|pf2Sex1| pf)))))))
```

———-

9.0.184 defun pfCollectArgTran

[pfCollect? p258]
[pf2sex1 p**??**]
[pfParts p247]
[pfCollectIterators p258]
[pfCollectBody p258]

— **defun pfCollectArgTran** —

```
(defun |pfCollectArgTran| (pf)
 (let (cond tmp2 tmp1 id conds)
  (cond
   ((|pfCollect?| pf)
    (setq conds (mapcar #'|pf2sex1| (|pfParts| (|pfCollectIterators| pf))))
    (setq id (|pf2Sex1| (|pfCollectBody| pf)))
    (cond
     ((and (consp conds)                    ; conds is [ ["|", cond] ]
           (eq (cdr conds) nil)
           (progn
            (setq tmp1 (car conds))
            (and (consp tmp1)
                 (eq (car tmp1) '|\||)
                 (progn
                  (setq tmp2 (cdr tmp1))
                  (and (consp tmp2)
                       (eq (cdr tmp2) nil)
                       (progn
                        (setq cond (car tmp2))
                        t))))))
       (list '|\|| id cond))
     (t (cons id conds))))
   (t (|pf2Sex1| pf)))))
```

———-

9.0.185 defun Convert a Lambda node to an S-expression

[pfLambdaTran p314]

— **defun pfLambda2Sex** —

```
(defun |pfLambda2Sex| (pf)
 (let (body argList tmp1)
  (setq tmp1 (|pfLambdaTran| pf))
  (setq argList (car tmp1))
  (setq body (cdr tmp1))
  (cons 'adef (cons argList body))))
```

———-

9.0.186 defun Convert a Rule node to an S-expression

[pfLhsRule2Sex p316]
[pfRuleLhsItems p284]
[pfRhsRule2Sex p317]
[pfRuleRhs p284]
[ruleLhsTran p320]
[rulePredicateTran p317]
[$multiVarPredicateList p??]
[$predicateList p??]
[$quotedOpList p??]

— **defun pfRule2Sex** —

```
(defun |pfRule2Sex| (pf)
 (let (|$multiVarPredicateList| |$predicateList| |$quotedOpList| rhs lhs)
 (declare (special |$multiVarPredicateList| |$predicateList| |$quotedOpList|))
  (setq |$quotedOpList| nil)
  (setq |$predicateList| nil)
  (setq |$multiVarPredicateList| nil)
  (setq lhs (|pfLhsRule2Sex| (|pfRuleLhsItems| pf)))
  (setq rhs (|pfRhsRule2Sex| (|pfRuleRhs| pf)))
  (setq lhs (|ruleLhsTran| lhs))
  (|rulePredicateTran|
   (if |$quotedOpList|
    (list '|rule| lhs rhs (cons '|construct| |$quotedOpList|))
    (list '|rule| lhs rhs)))))
```

———-

9.0.187 defun Convert the Lhs of a Rule to an S-expression

[pf2Sex1 p298]
[$insideRule p??]

— **defun pfLhsRule2Sex** —

```
(defun |pfLhsRule2Sex| (lhs)
 (let (|$insideRule|)
 (declare (special |$insideRule|))
  (setq |$insideRule| '|left|)
  (|pf2Sex1| lhs)))
```

———-

9.0.188 defun Convert the Rhs of a Rule to an S-expression

[pf2Sex1 p298]
[$insideRule p**??**]

— **defun pfRhsRule2Sex** —

```
(defun |pfRhsRule2Sex| (rhs)
 (let (|$insideRule|)
 (declare (special |$insideRule|))
  (setq |$insideRule| '|right|)
  (|pf2Sex1| rhs)))
```

———-

9.0.189 defun Convert a Rule predicate to an S-expression

```
;rulePredicateTran rule ==
;  null $multiVarPredicateList => rule
;  varList := patternVarsOf [rhs for [.,.,:rhs] in $multiVarPredicateList]
;  predBody :=
;    CDR $multiVarPredicateList =>
;      ['AND, :[:pvarPredTran(rhs, varList) for [.,.,:rhs] in
;        $multiVarPredicateList]]
;    [ [.,.,:rhs],:.] := $multiVarPredicateList
;    pvarPredTran(rhs, varList)
;  ['suchThat, rule,
;   ['construct, :[ ["QUOTE", var] for var in varList]],
;    ['ADEF, '(predicateVariable),
;     '((Boolean) (List (Expression (Integer)))), '(() ()),
;      predBody]]
```

[patternVarsOf p319]
[pvarPredTran p320]
[$multiVarPredicateList p**??**]

— **defun rulePredicateTran** —

```
(defun |rulePredicateTran| (rule)
 (let (predBody varList rhs tmp1 result)
 (declare (special |$multiVarPredicateList|))
  (if (null |$multiVarPredicateList|)
   rule
   (progn
   (setq varList
    (|patternVarsOf|
     ((lambda (t1 t2 t3)
      (loop
       (cond
        ((or (atom t2)
             (progn
              (setq t3 (car t2))
              nil))
           (return (nreverse t1)))
        (t
         (and (consp t3)
              (progn
               (setq tmp1 (cdr t3))
               (and (consp tmp1)
                    (progn
                     (setq rhs (cdr tmp1))
                     t)))
              (setq t1 (cons rhs t1)))))
       (setq t2 (cdr t2))))
      nil |$multiVarPredicateList| nil)))
   (setq predBody
    (cond
     ((cdr |$multiVarPredicateList|)
      (cons 'and
       ((lambda (t4 t5 t6)
        (loop
         (cond
          ((or (atom t5)
               (progn
                (setq t6 (car t5))
                nil))
            (return (nreverse t4)))
          (t
           (and (consp t6)
                (progn
                 (setq tmp1 (cdr t6))
                 (and (consp tmp1)
                      (progn
                       (setq rhs (cdr tmp1))
                       t)))
                      (setq t4
                       (append (reverse (|pvarPredTran| rhs varList))
                               t4)))))
```

```
        (setq t5 (cdr t5))))
       nil |$multiVarPredicateList| nil)))
    (t
     (progn
      (setq rhs (cddar |$multiVarPredicateList|))
      (|pvarPredTran| rhs varList)))))
  (dolist (var varList) (push (list 'quote var) result))
  (list '|suchThat| rule
   (cons '|construct| (nreverse result))
   (list 'adef '(|predicateVariable|)
                    '(((|Boolean|)
                      (|List| (|Expression| (|Integer|))))
                    '(nil nil) predBody))))))
```

———-

9.0.190 defun patternVarsOf

[patternVarsOf1 p319]

— **defun patternVarsOf** —

```
(defun |patternVarsOf| (expr)
 (|patternVarsOf1| expr nil))
```

———-

9.0.191 defun patternVarsOf1

[patternVarsOf1 p319]

— **defun patternVarsOf1** —

```
(defun |patternVarsOf1| (expr varList)
 (let (argl op)
  (cond
   ((null expr) varList)
   ((atom expr)
    (cond
     ((null (symbolp expr)) varList)
     ((member expr varList) varList)
     (t (cons expr varList))))
   ((and (consp expr)
         (progn
          (setq op (car expr))
```

```
          (setq argl (cdr expr))
          t))
         (progn
           (dolist (arg argl)
            (setq varList (|patternVarsOf1| arg varList)))
            varList))
   (t varList))))
```

———-

9.0.192 defun pvarPredTran

— **defun pvarPredTran** —

```
(defun |pvarPredTran| (rhs varList)
 (let ((i 0))
  (dolist (var varList rhs)
   (setq rhs (nsubst (list '|elt| '|predicateVariable| (incf i)) var rhs)))))
```

———-

9.0.193 defun Convert the Lhs of a Rule node to an S-expression

[patternVarsOf p319]
[nsubst p**??**]
[$predicateList p**??**]
[$multiVarPredicateList p**??**]

— **defun ruleLhsTran** —

```
(defun |ruleLhsTran| (ruleLhs)
 (let (predicate var vars predRhs predLhs name)
 (declare (special |$predicateList| |$multiVarPredicateList|))
 (dolist (pred |$predicateList|)
  (setq name (car pred))
  (setq predLhs (cadr pred))
  (setq predRhs (cddr pred))
  (setq vars (|patternVarsOf| predRhs))
  (cond
   ((cdr vars)
    (setq ruleLhs (nsubst predLhs name ruleLhs))
    (setq |$multiVarPredicateList| (cons pred |$multiVarPredicateList|)))
   (t
    (setq var (cadr predLhs))
```

```
    (setq predicate
      (list '|suchThat| predLhs (list 'adef (list var)
            '((|Boolean|) (|Expression| (|Integer|))) '(nil nil) predRhs)))
    (setq ruleLhs (nsubst predicate name ruleLhs)))))
  ruleLhs))
```

———-

9.0.194 defvar $dotdot

— **initvars** —

```
(defvar |$dotdot| '|..|)
```

———-

9.0.195 defun Translate ops into internal symbols

[$dotdot p321]

— **defun opTran 0** —

```
(defun |opTran| (op)
 (declare (special |$dotdot|))
 (cond
  ((equal op |$dotdot|) 'segment)
  ((eq op '[]) '|construct|)
  ((eq op '{}) '|braceFromCurly|)
  ((eq op 'is) '|is|)
  (t op)))
```

———-

Chapter 10

Keyed Message Handling

Throughout the interpreter there are messages printed using a symbol for a database lookup. This was done to enable translation of these messages languages other than English.

Axiom messages are read from a flat file database and returned as one long string. They are preceded in the database by a key and this is how they are referenced from code. For example, one key is S2IL0001 which means:

```
S2          Scratchpad II designation
I           from the interpreter
L           originally from LISPLIB BOOT
0001        a sequence number
```

Each message may contain formatting codes and and parameter codes. The formatting codes are:

```
%b          turn on bright printing
%ceoff      turn off centering
%ceon       turn on centering
%d          turn off bright printing
%f          user defined printing
%i          start indentation of 3 more spaces
%l          start a new line
%m          math-print an expression
%rjoff      turn off right justification (actually ragged left)
%rjon       turn on right justification (actually ragged left)
%s          pretty-print as an S-expression
%u          unindent 3 spaces
%x#         insert # spaces
```

The parameter codes look like %1, %2b, %3p, %4m, %5bp, %6s where the digit is the parameter number and the letters following indicate additional formatting. You can indicate as many additional formatting qualifiers as you like, to the degree they make sense.

- The “p” code means to call prefix2String on the parameter, a standard way of printing abbreviated types.
- The “P” operator maps prefix2String over its arguments.
- The “o” operation formats the argument as an operation name.
- The “b” means to print that parameter in a bold (bright) font.
- The “c” means to center that parameter on a new line.
- The “r” means to right justify (ragged left) the argument.
- The “f” means that the parameter is a list [fn, :args] and that “fn” is to be called on “args” to get the text.

Look in the file with the name defined in `$defaultMsgDatabaseName` above for examples.

10.0.196 defvar $cacheMessages

This is used for debugging

— **initvars** —

```
(defvar |$cacheMessages| t)
```

———-

10.0.197 defvar $msgAlist

— **initvars** —

```
(defvar |$msgAlist| nil)
```

———-

10.0.198 defvar $msgDatabaseName

— **initvars** —

```
(defvar |$msgDatabaseName| nil)
```

———-

10.0.199 defvar $testingErrorPrefix

— **initvars** —

```
(defvar |$testingErrorPrefix| "Daly Bug")
```

———-

10.0.200 defvar $texFormatting

— **initvars** —

```
(defvar |$texFormatting| nil)
```

———-

10.0.201 defvar $*msghash*

— **initvars** —

```
(defvar *msghash* nil "hash table keyed by msg number")
```

———-

10.0.202 defvar $msgdbPrims

— **initvars** —

```
(defvar |$msgdbPrims|
 '(|%b| |%d| |%l| |%i| |%u| %U |%n| |%x| |%ce| |%rj| "%U" "%b" "%d"
   "%l" "%i" "%u" "%U" "%n" "%x" "%ce" "%rj"))
```

———-

10.0.203 defvar $msgdbPunct

— **initvars** —

```
(defvar |$msgdbPunct|
  '(|.| |,| ! |:| |;| ? ] |)| "." "," "!" ":" ";" "?" "]" ")"))
```

———-

10.0.204 defvar $msgdbNoBlanksBeforeGroup

— **initvars** —

```
(defvar |$msgdbNoBlanksBeforeGroup|
 `(" " | | "%" % ,@|$msgdbPrims| ,@|$msgdbPunct|))
```

———-

10.0.205 defvar $msgdbNoBlanksAfterGroup

— **initvars** —

```
(defvar |$msgdbNoBlanksAfterGroup|
 `(" " | | "%" % ,@|$msgdbPrims| [ |(| "[" "("))
```

———-

10.0.206 defun Fetch a message from the message database

If the `*msghash*` hash table is empty we call `cacheKeyedMsg` to fill the table, otherwise we do a key lookup in the hash table. [object2Identifier p**??**]
[cacheKeyedMsg p327]
[$defaultMsgDatabaseName p8]
[*msghash* p325]

— **defun fetchKeyedMsg** —

```
(defun |fetchKeyedMsg| (key ignore)
 (declare (ignore ignore) (special *msghash* |$defaultMsgDatabaseName|))
 (setq key (|object2Identifier| key))
 (unless *msghash*
  (setq *msghash* (make-hash-table))
  (cacheKeyedMsg |$defaultMsgDatabaseName|))
 (gethash key *msghash*))
```

———-

10.0.207 defun Cache messages read from message database

[done p??]
[done p??]
[*msghash* p325]

— **defun cacheKeyedMsg** —

```
(defun cacheKeyedMsg (file)
 (let ((line "") (msg "") key)
 (declare (special *msghash*))
  (with-open-file (in file)
   (catch 'done
    (loop
     (setq line (read-line in nil nil))
     (cond
      ((null line)
       (when key (setf (gethash key *msghash*) msg))
       (throw 'done nil))
      ((= (length line) 0))
      ((char= (schar line 0) #\S)
       (when key (setf (gethash key *msghash*) msg))
       (setq key (intern line "BOOT"))
       (setq msg ""))
      ('else
       (setq msg (concatenate 'string msg line)))))))))
```

———-

10.0.208 defun getKeyedMsg

[fetchKeyedMsg p326]

— **defun getKeyedMsg** —

```
(defun |getKeyedMsg| (key) (|fetchKeyedMsg| key nil))
```

———-

10.0.209 defun Say a message using a keyed lookup

[sayKeyedMsgLocal p328]
[$texFormatting p325]

— **defun sayKeyedMsg** —

```
(defun |sayKeyedMsg| (key args)
 (let (|$texFormatting|)
 (declare (special |$texFormatting|))
  (setq |$texFormatting| nil)
  (|sayKeyedMsgLocal| key args)))
```

————-

10.0.210 defun Handle msg formatting and print to file

[segmentKeyedMsg p328]
[getKeyedMsg p327]
[substituteSegmentedMsg p**??**]
[flowSegmentedMsg p**??**]
[sayMSG2File p329]
[sayMSG p329]
[$printMsgsToFile p731]
[$linelength p765]
[$margin p765]
[$displayMsgNumber p737]

— **defun sayKeyedMsgLocal** —

```
(defun |sayKeyedMsgLocal| (key args)
 (let (msg msgp)
 (declare (special |$printMsgsToFile| $linelength $margin |$displayMsgNumber|))
  (setq msg (|segmentKeyedMsg| (|getKeyedMsg| key)))
  (setq msg (|substituteSegmentedMsg| msg args))
  (when |$displayMsgNumber| (setq msg `("%b" ,key |:| "%d" . ,msg)))
  (setq msgp (|flowSegmentedMsg| msg $linelength $margin))
  (when |$printMsgsToFile| (|sayMSG2File| msgp))
  (|sayMSG| msgp)))
```

————-

10.0.211 defun Break a message into words

[string2Words p**??**]

— **defun segmentKeyedMsg** —

```
(defun |segmentKeyedMsg| (msg) (|string2Words| msg))
```

———-

10.0.212 defun Write a msg into spadmsg.listing file

[makePathname p1036]
[defiostream p972]
[sayBrightly1 p1043]
[shut p972]

— **defun sayMSG2File** —

```
(defun |sayMSG2File| (msg)
 (let (file str)
  (setq file (|makePathname| '|spadmsg| '|listing| 'a))
  (setq str (defiostream '((mode . output) (file . ,file)) 255 0))
  (sayBrightly1 msg str)
  (shut str)))
```

———-

10.0.213 defun sayMSG

[saybrightly1 p**??**]
[$algebraOutputStream p754]

— **defun sayMSG** —

```
(defun |sayMSG| (x)
 (declare (special |$algebraOutputStream|))
 (when x (sayBrightly1 x |$algebraOutputStream|)))
```

———-

Chapter 11

Stream Utilities

The input stream is parsed into a large s-expression by repeated calls to Delay. Delay takes a function f and an argument x and returns a list consisting of `("nonnullstream" f x)`. Eventually multiple calls are made and a large list structure is created that consists of `("nonnullstream" f x ("nonnullstream" f1 x1 ("nonnullstream" f2 x2...`

This delay structure is given to StreamNull which walks along the list looking at the head. If the head is "nonnullstream" then the function is applied to the argument.

So, in effect, the input is "zipped up" into a Delay data structure which is then evaluated by calling StreamNull. This "zippered stream" parser was a research project at IBM and Axiom was the testbed (which explains the strange parsing technique).

11.0.214 defun npNull

[StreamNull p331]

— **defun npNull** —

```
(defun |npNull| (x) (|StreamNull| x))
```

———-

11.0.215 defun StreamNull

[eqcar p??]

— **defun StreamNull 0** —

```
(defun |StreamNull| (x)
```

```
(let (st)
 (cond
  ((or (null x) (eqcar x '|nullstream|)) t)
  (t
   ((lambda nil
    (loop
     (cond
      ((not (eqcar x '|nonnullstream|)) (return nil))
      (t
       (setq st (apply (cadr x) (cddr x)))
       (rplaca x (car st))
       (rplacd x (cdr st)))))))
    (eqcar x '|nullstream|)))))
```

Chapter 12

Code Piles

The insertpiles function converts a line-list to a line-forest where a line is a token-dequeue and has a column which is an integer. An A-forest is an A-tree-list An A-tree has a root which is an A, and subtrees which is an A-forest.

A forest with more than one tree corresponds to a Scratchpad pile structure (t1;t2;t3;...;tn), and a tree corresponds to a pile item. The (; and) tokens are inserted into a ¿1-forest, otherwise the root of the first tree is concatenated with its forest. column t is the number of spaces before the first non-space in line t

12.0.216 defun insertpile

[npNull p331]
[pilePlusComment p334]
[pilePlusComments p334]
[pileTree p335]
[pileCforest p338]

— defun insertpile —

```
(defun |insertpile| (s)
 (let (stream a t1 h1 t2 h tmp1)
  (cond
   ((|npNull| s) (list nil 0 nil s))
   (t
    (setq tmp1 (list (car s) (cdr s)))
    (setq h (car tmp1))
    (setq t2 (cadr tmp1))
    (cond
     ((|pilePlusComment| h)
       (setq tmp1 (|pilePlusComments| s))
       (setq h1 (car tmp1))
```

```
       (setq t1 (cadr tmp1))
       (setq a (|pileTree| (- 1) t1))
       (cons (list (|pileCforest|
                      (append h1 (cons (elt a 2) nil))))
             (elt a 3)))
      (t
       (setq stream (cadar s))
       (setq a (|pileTree| -1 s))
       (cons (list (list (elt a 2) stream)) (elt a 3))))))))
```

———-

12.0.217 defun pilePlusComment

[tokType p411]
[npNull p331]
[pilePlusComment p334]
[pilePlusComments p334]

— **defun pilePlusComment** —

```
(defun |pilePlusComment| (arg)
 (eq (|tokType| (caar arg)) '|comment|))
```

———-

12.0.218 defun pilePlusComments

— **defun pilePlusComments** —

```
(defun |pilePlusComments| (s)
 (let (t1 h1 t2 h tmp1)
  (cond
   ((|npNull| s) (list nil s))
   (t
    (setq tmp1 (list (car s) (cdr s)))
    (setq h (car tmp1))
    (setq t2 (cadr tmp1))
    (cond
     ((|pilePlusComment| h)
      (setq tmp1 (|pilePlusComments| t2))
      (setq h1 (car tmp1))
      (setq t1 (cadr tmp1))
      (list (cons h h1) t1))
```

```
    (t
     (list nil s)))))))
```

———-

12.0.219 defun pileTree

[npNull p331]
[pileColumn p335]
[pileForests p335]

— **defun pileTree** —

```
(defun |pileTree| (n s)
 (let (hh t1 h tmp1)
  (cond
   ((|npNull| s) (list nil n nil s))
   (t
    (setq tmp1 (list (car s) (cdr s)))
    (setq h (car tmp1))
    (setq t1 (cadr tmp1))
    (setq hh (|pileColumn| (car h)))
    (cond
     ((< n hh) (|pileForests| (car h) hh t1))
     (t (list nil n nil s)))))))
```

———-

12.0.220 defun pileColumn

[tokPosn p411]

— **defun pileColumn** —

```
(defun |pileColumn| (arg)
 (cdr (|tokPosn| (caar arg))))
```

———-

12.0.221 defun pileForests

[pileForest p336]
[npNull p331]

[pileForests p335]
[pileCtree p338]

— **defun pileForests** —

```
(defun |pileForests| (h n s)
 (let (t1 h1 tmp1)
  (setq tmp1 (|pileForest| n s))
  (setq h1 (car tmp1))
  (setq t1 (cadr tmp1))
  (cond
   ((|npNull| h1) (list t n h s))
   (t (|pileForests| (|pileCtree| h h1) n t1)))))
```

———-

12.0.222 defun pileForest

[pileTree p335]
[pileForest1 p336]

— **defun pileForest** —

```
(defun |pileForest| (n s)
 (let (t1 h1 t2 h hh b tmp)
  (setq tmp (|pileTree| n s))
  (setq b (car tmp))
  (setq hh (cadr tmp))
  (setq h (caddr tmp))
  (setq t2 (cadddr tmp))
  (cond
   (b
    (setq tmp (|pileForest1| hh t2))
    (setq h1 (car tmp))
    (setq t1 (cadr tmp))
    (list (cons h h1) t1))
   (t
    (list nil s)))))
```

———-

12.0.223 defun pileForest1

[eqpileTree p337]
[pileForest1 p336]

— **defun pileForest1** —

```
(defun |pileForest1| (n s)
 (let (t1 h1 t2 h n1 b tmp)
  (setq tmp (|eqpileTree| n s))
  (setq b (car tmp))
  (setq n1 (cadr tmp))
  (setq h (caddr tmp))
  (setq t2 (cadddr tmp))
  (cond
   (b
    (setq tmp (|pileForest1| n t2))
    (setq h1 (car tmp))
    (setq t1 (cadr tmp))
    (list (cons h h1) t1))
   (t (list nil s)))))
```

———-

12.0.224 defun eqpileTree

[npNull p331]
[pileColumn p335]
[pileForests p335]

— **defun eqpileTree** —

```
(defun |eqpileTree| (n s)
 (let (hh t1 h tmp)
  (cond
   ((|npNull| s) (list nil n nil s))
   (t
    (setq tmp (list (car s) (cdr s)))
    (setq h (car tmp))
    (setq t1 (cadr tmp))
    (setq hh (|pileColumn| (car h)))
    (cond
     ((equal hh n) (|pileForests| (car h) hh t1))
     (t (list nil n nil s)))))))
```

———-

12.0.225 defun pileCtree

[dqAppend p342]
[pileCforest p338]

— **defun pileCtree** —

```
(defun |pileCtree| (x y)
 (|dqAppend| x (|pileCforest| y)))
```

————-

12.0.226 defun pileCforest

Only enpiles forests with >= 2 trees [tokPart p411]
[enPile p338]
[separatePiles p339]

— **defun pileCforest** —

```
(defun |pileCforest| (x)
 (let (f)
  (cond
   ((null x) nil)
   ((null (cdr x)) (setq f (car x))
    (cond
     ((eq (|tokPart| (caar f)) 'if) (|enPile| f))
     (t f)))
   (t (|enPile| (|separatePiles| x))))))
```

————-

12.0.227 defun enPile

[dqConcat p341]
[dqUnit p341]
[tokConstruct p409]
[firstTokPosn p339]
[lastTokPosn p339]

— **defun enPile** —

```
(defun |enPile| (x)
 (|dqConcat|
```

```
  (list
   (|dqUnit| (|tokConstruct| '|key| 'settab (|firstTokPosn| x)))
   x
   (|dqUnit| (|tokConstruct| '|key| 'backtab (|lastTokPosn| x))))))
```

———-

12.0.228 defun firstTokPosn

[tokPosn p411]

— **defun firstTokPosn** —

```
(defun |firstTokPosn| (arg) (|tokPosn| (caar arg)))
```

———-

12.0.229 defun lastTokPosn

[tokPosn p411]

— **defun lastTokPosn** —

```
(defun |lastTokPosn| (arg) (|tokPosn| (cadr arg)))
```

———-

12.0.230 defun separatePiles

[dqUnit p341]
[tokConstruct p409]
[lastTokPosn p339]
[dqConcat p341]
[separatePiles p339]

— **defun separatePiles** —

```
(defun |separatePiles| (x)
 (let (semicolon a)
  (cond
   ((null x) nil)
   ((null (cdr x)) (car x))
```

```
(t
 (setq a (car x))
 (setq semicolon
   (|dqUnit| (|tokConstruct| '|key| 'backset (|lastTokPosn| a))))
 (|dqConcat| (list a semicolon (|separatePiles| (cdr x)))))))
```

———-

Chapter 13

Dequeue Functions

The dqUnit makes a unit dq i.e. a dq with one item, from the item

13.0.231 defun dqUnit

— **defun dqUnit 0** —

```
(defun |dqUnit| (s)
 (let (a)
  (setq a (list s))
  (cons a a)))
```

————-

13.0.232 defun dqConcat

The dqConcat function concatenates a list of dq's, destroying all but the last [dqAppend p342]
[dqConcat p341]

— **defun dqConcat** —

```
(defun |dqConcat| (ld)
 (cond
  ((null ld) nil)
  ((null (cdr ld)) (car ld))
  (t (|dqAppend| (car ld) (|dqConcat| (cdr ld))))))
```

————-

13.0.233 defun dqAppend

The dqAppend function appends 2 dq's, destroying the first

— **defun dqAppend 0** —

```
(defun |dqAppend| (x y)
 (cond
  ((null x) y)
  ((null y) x)
  (t
   (rplacd (cdr x) (car y))
   (rplacd x (cdr y)) x)))
```

————-

13.0.234 defun dqToList

— **defun dqToList 0** —

```
(defun |dqToList| (s)
 (when s (car s)))
```

————-

Chapter 14

Message Handling

14.1 The Line Object

14.1.1 defun Line object creation

This is called in only one place, the incLine1 function.

— **defun lnCreate 0** —

```
(defun |lnCreate| (extraBlanks string globalNum &rest optFileStuff)
 (let ((localNum (first optFileStuff))
       (filename (second optFileStuff)))
  (unless localNum (setq localNum 0))
  (list extraBlanks string globalNum localNum filename)))
```

———-

14.1.2 defun Line element 0; Extra blanks

— **defun lnExtraBlanks 0** —

```
(defun |lnExtraBlanks| (lineObject) (elt lineObject 0))
```

———-

14.1.3 defun Line element 1; String

— **defun lnString 0** —

```
(defun |lnString| (lineObject) (elt lineObject 1))
```

———-

14.1.4 defun Line element 2; Globlal number

— **defun lnGlobalNum 0** —

```
(defun |lnGlobalNum| (lineObject) (elt lineObject 2))
```

———-

14.1.5 defun Line element 2; Set Global number

— **defun lnSetGlobalNum 0** —

```
(defun |lnSetGlobalNum| (lineObject num)
 (setf (elt lineObject 2) num))
```

———-

14.1.6 defun Line elemnt 3; Local number

— **defun lnLocalNum 0** —

```
(defun |lnLocalNum| (lineObject) (elt lineObject 3))
```

———-

14.1.7 defun Line element 4; Place of origin

— **defun lnPlaceOfOrigin 0** —

```
(defun |lnPlaceOfOrigin| (lineObject) (elt lineObject 4))
```

———-

14.1.8 defun Line element 4: Is it a filename?

[lnFileName? p345]

— **defun lnImmediate? 0** —

```
(defun |lnImmediate?| (lineObject) (null (|lnFileName?| lineObject)))
```

————-

14.1.9 defun Line element 4: Is it a filename?

— **defun lnFileName? 0** —

```
(defun |lnFileName?| (lineObject)
 (let (filename)
  (when (consp (setq filename (elt lineObject 4))) filename)))
```

————-

14.1.10 defun Line element 4; Get filename

[lnFileName? p345]
[ncBug p366]

— **defun lnFileName** —

```
(defun |lnFileName| (lineObject)
 (let (fN)
  (if (setq fN (|lnFileName?| lineObject))
   fN
   (|ncBug| "there is no file name in %1" (list lineObject)))))
```

————-

14.2 Messages

14.2.1 defun msgCreate

```
msgObject  tag -- catagory of msg
                  -- attributes as a-list
```

```
                    'imPr  => dont save for list processing
                    toWhere, screen or file
                    'norep => only display once in list
          pos -- position with possible FROM/TO tag
          key -- key for message database
         argL -- arguments to be placed in the msg test
       prefix -- things like "Error: "
         text -- the actual text
```

[setMsgForcedAttrList p362]
[putDatabaseStuff p363]
[initImPr p365]
[initToWhere p366]

— **defun msgCreate** —

```
(defun |msgCreate| (tag posWTag key argL optPre &rest optAttr)
 (let (msg)
  (when (consp key) (setq tag '|old|))
  (setq msg (list tag posWTag key argL optPre nil))
  (when (car optAttr) (|setMsgForcedAttrList| msg (car optAttr)))
  (|putDatabaseStuff| msg)
  (|initImPr| msg)
  (|initToWhere| msg)
  msg))
```

————-

14.2.2 defun getMsgPosTagOb

— **defun getMsgPosTagOb 0** —

```
(defun |getMsgPosTagOb| (msg) (elt msg 1))
```

————-

14.2.3 defun getMsgKey

— **defun getMsgKey 0** —

```
(defun |getMsgKey| (msg)  (elt msg 2))
```

————-

14.2.4 defun getMsgArgL

— **defun getMsgArgL 0** —

```
(defun |getMsgArgL| (msg) (elt msg 3))
```

————-

14.2.5 defun getMsgPrefix

— **defun getMsgPrefix 0** —

```
(defun |getMsgPrefix| (msg) (elt msg 4))
```

————-

14.2.6 defun setMsgPrefix

— **defun setMsgPrefix 0** —

```
(defun |setMsgPrefix| (msg val) (setf (elt msg 4) val))
```

————-

14.2.7 defun getMsgText

— **defun getMsgText 0** —

```
(defun |getMsgText| (msg) (elt msg 5))
```

————-

14.2.8 defun setMsgText

— **defun setMsgText 0** —

```
(defun |setMsgText| (msg val)
  (setf (elt msg 5) val))
```

———-

14.2.9 defun getMsgPrefix?

— **defun getMsgPrefix? 0** —

```
(defun |getMsgPrefix?| (msg)
 (let ((pre (|getMsgPrefix| msg)))
  (unless (eq pre '|noPre|) pre)))
```

———-

14.2.10 defun getMsgTag

The valid message tags are: line, old, error, warn, bug, unimple, remark, stat, say, debug [ncTag p413]

— **defun getMsgTag 0** —

```
(defun |getMsgTag| (msg) (|ncTag| msg))
```

———-

14.2.11 defun getMsgTag?

[IFCAR p**??**]
[getMsgTag p348]

— **defun getMsgTag? 0** —

```
(defun |getMsgTag?| (|msg|)
 (ifcar (member (|getMsgTag| |msg|)
         (list '|line| '|old| '|error| '|warn| '|bug|
               '|unimple| '|remark| '|stat| '|say| '|debug|))))
```

———-

14.2.12 defun line?

[getMsgTag p348]

— **defun line?** —

```
(defun |line?| (msg) (eq (|getMsgTag| msg) '|line|))
```

————-

14.2.13 defun leader?

[getMsgTag p348]

— **defun leader?** —

```
(defun |leader?| (msg) (eq (|getMsgTag| msg) '|leader|))
```

————-

14.2.14 defun toScreen?

[getMsgToWhere p361]

— **defun toScreen?** —

```
(defun |toScreen?| (msg) (not (eq (|getMsgToWhere| msg) '|fileOnly|)))
```

————-

14.2.15 defun ncSoftError

Messages for the USERS of the compiler. The program being compiled has a minor error. Give a message and continue processing. [desiredMsg p350]
[processKeyedError p351]
[msgCreate p345]
[$newcompErrorCount p28]

— **defun ncSoftError** —

```
(defun |ncSoftError| (pos erMsgKey erArgL &rest optAttr)
 (declare (special |$newcompErrorCount|))
```

```
  (setq |$newcompErrorCount| (+ |$newcompErrorCount| 1))
  (when (|desiredMsg| erMsgKey)
   (|processKeyedError|
    (|msgCreate| '|error| pos erMsgKey erArgL
                 "Error" optAttr))))
```

———-

14.2.16 defun ncHardError

The program being compiled is seriously incorrect. Give message and throw to a recovery point. [desiredMsg p350]
[processKeyedError p351]
[msgCreate p345]
[ncError p69]
[$newcompErrorCount p28]

— **defun ncHardError** —

```
(defun |ncHardError| (pos erMsgKey erArgL &rest optAttr)
 (let (erMsg)
 (declare (special |$newcompErrorCount|))
  (setq |$newcompErrorCount| (+ |$newcompErrorCount| 1))
  (if (|desiredMsg| erMsgKey)
   (setq erMsg
    (|processKeyedError|
     (|msgCreate| '|error| pos erMsgKey erArgL "Error" optAttr)))
   (|ncError|))))
```

———-

14.2.17 defun desiredMsg

— **defun desiredMsg 0** —

```
(defun |desiredMsg| (erMsgKey &rest optCatFlag)
 (declare (ignore erMsgKey))
 (cond
  ((null (null optCatFlag)) (car optCatFlag))
  (t t)))
```

———-

14.2.18 defun processKeyedError

[getMsgTag? p348]
[getMsgKey p346]
[getMsgPrefix? p348]
[sayBrightly p??]
[CallerName p??]
[msgImPr? p356]
[msgOutputter p351]
[$ncMsgList p27]

— **defun processKeyedError** —

```
(defun |processKeyedError| (msg)
 (prog (pre erMsg)
 (declare (special |$ncMsgList|))
  (cond
   ((eq (|getMsgTag?| msg) '|old|)
     (setq erMsg (|getMsgKey| msg))
     (cond
      ((setq pre (|getMsgPrefix?| msg))
        (setq erMsg (cons '|%b| (cons pre (cons '|%d| erMsg))))))
     (|sayBrightly| (cons "old msg from " (cons (|CallerName| 4) erMsg))))
   ((|msgImPr?| msg) (|msgOutputter| msg))
   (t (setq |$ncMsgList| (cons msg |$ncMsgList|))))))
```

————-

14.2.19 defun msgOutputter

[getStFromMsg p352]
[leader? p349]
[line? p349]
[toScreen? p349]
[flowSegmentedMsg p??]
[sayBrightly p??]
[toFile? p361]
[alreadyOpened? p361]
[$linelength p765]

— **defun msgOutputter** —

```
(defun |msgOutputter| (msg)
 (let (alreadyOpened shouldFlow st)
 (declare (special $linelength))
  (setq st (|getStFromMsg| msg))
```

```
  (setq shouldFlow (null (or (|leader?| msg) (|line?| msg))))
  (when (|toScreen?| msg)
    (when shouldFlow (setq st (|flowSegmentedMsg| st $linelength 0)))
    (|sayBrightly| st))
  (when (|toFile?| msg)
    (when shouldFlow (setq st (|flowSegmentedMsg| st (- $linelength 6) 0)))
    (setq alreadyOpened (|alreadyOpened?| msg)))))
```

———-

14.2.20 defun listOutputter

[msgOutputter p351]

— **defun listOutputter** —

```
(defun |listOutputter| (outputList)
 (dolist (msg outputList)
  (|msgOutputter| msg)))
```

———-

14.2.21 defun getStFromMsg

[getPreStL p353]
[getMsgPrefix? p348]
[getMsgTag p348]
[getMsgText p347]
[getPosStL p354]
[getMsgKey? p360]
[pname p1039]
[getMsgLitSym p360]
[tabbing p360]

— **defun getStFromMsg** —

```
(defun |getStFromMsg| (msg)
 (let (st posStL preStL)
  (setq preStL (|getPreStL| (|getMsgPrefix?| msg)))
  (cond
   ((eq (|getMsgTag| msg) '|line|)
     (cons ""
       (cons "%x1" (append preStL (cons (|getMsgText| msg) nil)))))
   (t
```

```
    (setq posStL (|getPosStL| msg))
    (setq st
     (cons posStL
      (cons (|getMsgLitSym| msg)
       (cons ""
        (append preStL
         (cons (|tabbing| msg)
          (|getMsgText| msg)))))))))))
```

———-

14.2.22 defvar $preLength

— **initvars** —

```
(defvar |$preLength| 11)
```

———-

14.2.23 defun getPreStL

[size p1039]
[$preLength p353]

— **defun getPreStL 0** —

```
(defun |getPreStL| (optPre)
 (let (spses extraPlaces)
 (declare (special |$preLength|))
  (cond
   ((null optPre) (list "  "))
   (t
    (setq spses
     (cond
      ((< 0 (setq extraPlaces (- (- |$preLength| (size optPre)) 3)))
       (make-string extraPlaces))
      (t ""))))
    (list '|%b| optPre spses ":" '|%d|)))))
```

———-

14.2.24 defun getPosStL

[showMsgPos? p355]
[getMsgPos p357]
[msgImPr? p356]
[decideHowMuch p357]
[listDecideHowMuch p359]
[ppos p355]
[remLine p360]
[remFile p355]
[$lastPos p??]

— defun getPosStL —

```
(defun |getPosStL| (msg)
 (let (printedOrigin printedLineNum printedFileName fullPrintedPos howMuch
       msgPos)
 (declare (special |$lastPos|))
 (cond
  ((null (|showMsgPos?| msg)) "")
  (t
   (setq msgPos (|getMsgPos| msg))
   (setq howMuch
    (if (|msgImPr?| msg)
     (|decideHowMuch| msgPos |$lastPos|)
     (|listDecideHowMuch| msgPos |$lastPos|)))
   (setq |$lastPos| msgPos)
   (setq fullPrintedPos (|ppos| msgPos))
   (setq printedFileName
    (cons "%x2" (cons "[" (append (|remLine| fullPrintedPos) (cons "]" nil)))))
   (setq printedLineNum
    (cons "%x2" (cons "[" (append (|remFile| fullPrintedPos) (cons "]" nil)))))
   (setq printedOrigin
    (cons "%x2" (cons "[" (append fullPrintedPos (cons "]" nil)))))
   (cond
    ((eq howMuch 'org)
     (cons "" (append printedOrigin (cons '|%l| nil))))
    ((eq howMuch 'line)
     (cons "" (append printedLineNum (cons '|%l| nil))))
    ((eq howMuch 'file)
     (cons "" (append printedFileName (cons '|%l| nil))))
    ((eq howMuch 'all)
     (cons ""
      (append printedFileName
       (cons '|%l|
        (cons ""
         (append printedLineNum
          (cons '|%l| nil)))))))
    (t ""))))))
```

———-

14.2.25 defun ppos

[pfNoPosition? p410]
[pfImmediate? p??]
[pfCharPosn p233]
[pfLinePosn p233]
[porigin p88]
[pfFileName p234]

— **defun ppos** —

```
(defun |ppos| (p)
 (let (org lpos cpos)
  (cond
   ((|pfNoPosition?| p) (list "no position"))
   ((|pfImmediate?| p) (list "console"))
   (t
    (setq cpos (|pfCharPosn| p))
    (setq lpos (|pfLinePosn| p))
    (setq org (|porigin| (|pfFileName| p)))
    (list org " " "line" " " lpos)))))
```

———-

14.2.26 defun remFile

[IFCDR p??]
[IFCAR p??]

— **defun remFile** —

```
(defun |remFile| (positionList) (ifcdr (ifcdr positionList)))
```

———-

14.2.27 defun showMsgPos?

[msgImPr? p356]
[leader? p349]

[$erMsgToss p??]

— **defun showMsgPos? 0** —

```
(defun |showMsgPos?| (msg)
 (declare (special |$erMsgToss|))
 (or |$erMsgToss| (and (null (|msgImPr?| msg)) (null (|leader?| msg)))))
```

————-

14.2.28 defvar $imPrGuys

— **initvars** —

```
(defvar |$imPrGuys| (list '|imPr|))
```

————-

14.2.29 defun msgImPr?

[getMsgCatAttr p356]

— **defun msgImPr?** —

```
(defun |msgImPr?| (msg)
 (eq (|getMsgCatAttr| msg '|$imPrGuys|) '|imPr|))
```

————-

14.2.30 defun getMsgCatAttr

[ifcdr p??]
[qassq p??]
[ncAlist p413]

— **defun getMsgCatAttr** —

```
(defun |getMsgCatAttr| (msg cat)
 (ifcdr (qassq cat (|ncAlist| msg))))
```

————-

14.2.31 defun getMsgPos

[getMsgFTTag? p357]
[getMsgPosTagOb p346]

— **defun getMsgPos** —

```
(defun |getMsgPos| (msg)
 (if (|getMsgFTTag?| msg)
  (cadr (|getMsgPosTagOb| msg))
  (|getMsgPosTagOb| msg)))
```

———-

14.2.32 defun getMsgFTTag?

[ifcar p**??**]
[getMsgPosTagOb p346]

— **defun getMsgFTTag?** —

```
(defun |getMsgFTTag?| (msg)
 (ifcar (member (ifcar (|getMsgPosTagOb| msg)) (list 'from 'to 'fromto))))
```

———-

14.2.33 defun decideHowMuch

When printing a msg, we wish not to show pos information that was shown for a previous msg with identical pos info. org prints out the word noposition or console [poNopos? p358]
[poPosImmediate? p358]
[poFileName p358]
[poLinePosn p359]

— **defun decideHowMuch** —

```
(defun |decideHowMuch| (pos oldPos)
 (cond
  ((or (and (|poNopos?| pos) (|poNopos?| oldPos))
       (and (|poPosImmediate?| pos) (|poPosImmediate?| oldPos)))
    'none)
  ((or (|poNopos?| pos) (|poPosImmediate?| pos)) 'org)
  ((or (|poNopos?| oldPos) (|poPosImmediate?| oldPos)) 'all)
  ((not (equal (|poFileName| oldPos) (|poFileName| pos))) 'all)
```

```
  ((not (equal (|poLinePosn| oldPos) (|poLinePosn| pos))) 'line)
  (t 'none)))
```

———-

14.2.34 defun poNopos?

— **defun poNopos? 0** —

```
(defun |poNopos?| (posn)
 (equal posn (list '|noposition|)))
```

———-

14.2.35 defun poPosImmediate?

[poNopos? p358]
[lnImmediate? p345]
[poGetLineObject p359]

— **defun poPosImmediate?** —

```
(defun |poPosImmediate?| (txp)
 (unless (|poNopos?| txp) (|lnImmediate?| (|poGetLineObject| txp))))
```

———-

14.2.36 defun poFileName

[lnFileName p345]
[poGetLineObject p359]

— **defun poFileName** —

```
(defun |poFileName| (posn)
 (if posn
  (|lnFileName| (|poGetLineObject| posn))
  (caar posn)))
```

———-

14.2.37 defun poGetLineObject

— **defun poGetLineObject 0** —

```
(defun |poGetLineObject| (posn)
 (car posn))
```

————-

14.2.38 defun poLinePosn

[lnLocalNum p344]
[poGetLineObject p359]

— **defun poLinePosn** —

```
(defun |poLinePosn| (posn)
 (if posn
  (|lnLocalNum| (|poGetLineObject| posn))
  (cdar posn)))
```

————-

14.2.39 defun listDecideHowMuch

[poNopos? p358]
[poPosImmediate? p358]
[poGlobalLinePosn p72]

— **defun listDecideHowMuch** —

```
(defun |listDecideHowMuch| (pos oldPos)
 (cond
  ((or (and (|poNopos?| pos) (|poNopos?| oldPos))
       (and (|poPosImmediate?| pos) (|poPosImmediate?| oldPos)))
    'none)
  ((|poNopos?| pos) 'org)
  ((|poNopos?| oldPos) 'none)
  ((< (|poGlobalLinePosn| pos) (|poGlobalLinePosn| oldPos))
   (if (|poPosImmediate?| pos) 'org 'line))
  (t 'none)))
```

————-

14.2.40 defun remLine

— **defun remLine 0** —

```
(defun |remLine| (positionList) (list (ifcar positionList)))
```

————-

14.2.41 defun getMsgKey?

[identp p1040]

— **defun getMsgKey? 0** —

```
(defun |getMsgKey?| (msg)
 (let ((val (|getMsgKey| msg)))
  (when (identp val) val)))
```

————-

14.2.42 defun getMsgLitSym

[getMsgKey? p360]

— **defun getMsgLitSym** —

```
(defun |getMsgLitSym| (msg)
 (if (|getMsgKey?| msg) " " "*"))
```

————-

14.2.43 defun tabbing

[getMsgPrefix? p348]
[$preLength p353]

— **defun tabbing** —

```
(defun |tabbing| (msg)
 (let (chPos)
 (declare (special |$preLength|))
```

```
  (setq chPos 2)
  (when (|getMsgPrefix?| msg) (setq chPos (- (+ chPos |$preLength|) 1)))
  (cons '|%t| chPos)))
```

———-

14.2.44 defvar $toWhereGuys

— **initvars** —

```
(defvar |$toWhereGuys| (list '|fileOnly| '|screenOnly|))
```

———-

14.2.45 defun getMsgToWhere

[getMsgCatAttr p356]

— **defun getMsgToWhere** —

```
(defun |getMsgToWhere| (msg) (|getMsgCatAttr| msg '|$toWhereGuys|))
```

———-

14.2.46 defun toFile?

[getMsgToWhere p361]
[$fn p**??**]

— **defun toFile?** —

```
(defun |toFile?| (msg)
 (declare (special |$fn|))
 (and (consp |$fn|) (not (eq (|getMsgToWhere| msg) '|screenOnly|))))
```

———-

14.2.47 defun alreadyOpened?

[msgImPr? p356]

— **defun alreadyOpened?** —

```
(defun |alreadyOpened?| (msg) (null (|msgImPr?| msg)))
```

———-

14.2.48 defun setMsgForcedAttrList

[setMsgForcedAttr p362]
[whichCat p363]

— **defun setMsgForcedAttrList** —

```
(defun |setMsgForcedAttrList| (msg attrlist)
 (dolist (attr attrlist)
  (|setMsgForcedAttr| msg (|whichCat| attr) attr)))
```

———-

14.2.49 defun setMsgForcedAttr

[setMsgCatlessAttr p363]
[ncPutQ p414]

— **defun setMsgForcedAttr** —

```
(defun |setMsgForcedAttr| (msg cat attr)
 (if (eq cat '|catless|)
  (|setMsgCatlessAttr| msg attr)
  (|ncPutQ| msg cat attr)))
```

———-

14.2.50 defvar $attrCats

— **initvars** —

```
(defvar |$attrCats| (list '|$imPrGuys| '|$toWhereGuys| '|$repGuys|))
```

———-

14.2.51 defun whichCat

[ListMember? p??]
[$attrCats p362]

— **defun whichCat** —

```
(defun |whichCat| (attr)
 (let ((found '|catless|) done)
 (declare (special |$attrCats|))
  (loop for cat in |$attrCats| do
    (when (|ListMember?| attr (eval cat))
     (setq found cat)
     (setq done t))
   until done)
  found))
```

———-

14.2.52 defun setMsgCatlessAttr

TPDHERE: Changed from —catless— to '—catless— [ncPutQ p414]
[ifcdr p??]
[qassq p??]
[ncAlist p413]

— **defun setMsgCatlessAttr** —

```
(defun |setMsgCatlessAttr| (msg attr)
 (|ncPutQ| msg '|catless|
  (cons attr (ifcdr (qassq |catless| (|ncAlist| msg))))))
```

———-

14.2.53 defun putDatabaseStuff

TPDHERE: The variable al is undefined [getMsgInfoFromKey p364]
[setMsgUnforcedAttrList p364]
[setMsgText p347]

— **defun putDatabaseStuff** —

```
(defun |putDatabaseStuff| (msg)
 (let (attributes text tmp)
```

```
  (setq tmp (|getMsgInfoFromKey| msg))
  (setq text (car tmp))
  (setq attributes (cadr tmp))
  (when attributes (|setMsgUnforcedAttrList| msg al))
  (|setMsgText| msg text)))
```

———-

14.2.54 defun getMsgInfoFromKey

[getMsgKey? p360]
[getErFromDbL p??]
[getMsgKey p346]
[segmentKeyedMsg p328]
[removeAttributes p??]
[substituteSegmentedMsg p??]
[getMsgArgL p347]
[$msgDatabaseName p324]

— defun getMsgInfoFromKey —

```
(defun |getMsgInfoFromKey| (msg)
 (let (|$msgDatabaseName| attributes tmp msgText msgKey)
 (declare (special |$msgDatabaseName|))
  (setq |$msgDatabaseName| nil)
  (setq msgText
   (cond
    ((setq msgKey (|getMsgKey?| msg))
     (|fetchKeyedMsg| msgKey nil))
    (t (|getMsgKey| msg))))
  (setq msgText (|segmentKeyedMsg| msgText))
  (setq tmp (|removeAttributes| msgText))
  (setq msgText (car tmp))
  (setq attributes (cadr tmp))
  (setq msgText (|substituteSegmentedMsg| msgText (|getMsgArgL| msg)))
  (list msgText attributes)))
```

———-

14.2.55 defun setMsgUnforcedAttrList

[setMsgUnforcedAttr p365]
[whichCat p363]

— defun setMsgUnforcedAttrList —

```
(defun |setMsgUnforcedAttrList| (msg attrlist)
 (dolist (attr attrlist)
  (|setMsgUnforcedAttr| msg (|whichCat| attr) attr)))
```

———-

14.2.56 defun setMsgUnforcedAttr

[setMsgCatlessAttr p363]
[qassq p??]
[ncAlist p413]
[ncPutQ p414]

— **defun setMsgUnforcedAttr** —

```
(defun |setMsgUnforcedAttr| (msg cat attr)
 (cond
  ((eq cat '|catless|) (|setMsgCatlessAttr| msg attr))
  ((null (qassq cat (|ncAlist| msg))) (|ncPutQ| msg cat attr))))
```

———-

14.2.57 defvar $imPrTagGuys

— **initvars** —

```
(defvar |$imPrTagGuys| (list '|unimple| '|bug| '|debug| '|say| '|warn|))
```

———-

14.2.58 defun initImPr

[getMsgTag p348]
[setMsgUnforcedAttr p365]
[$imPrTagGuys p365]
[$erMsgToss p??]

— **defun initImPr** —

```
(defun |initImPr| (msg)
 (declare (special |$imPrTagGuys| |$erMsgToss|))
```

```
 (when (or |$erMsgToss| (member (|getMsgTag| msg) |$imPrTagGuys|))
   (|setMsgUnforcedAttr| msg '|$imPrGuys| '|imPr|)))
```

———-

14.2.59 defun initToWhere

[getMsgCatAttr p356]
[setMsgUnforcedAttr p365]

— **defun initToWhere** —

```
(defun |initToWhere| (msg)
 (if (member '|trace| (|getMsgCatAttr| msg '|catless|))
   (|setMsgUnforcedAttr| msg '|$toWhereGuys| '|screenOnly|)))
```

———-

14.2.60 defun ncBug

Bug in the compiler: something which shouldn't have happened did. [processKeyedError p351]
[msgCreate p345]
[enable-backtrace p**??**]
[ncAbort p**??**]
[$nopos p28]
[$newcompErrorCount p28]

— **defun ncBug** —

```
(defun |ncBug| (erMsgKey erArgL &rest optAttr)
 (let (erMsg)
 (declare (special |$nopos| |$newcompErrorCount|))
  (setq |$newcompErrorCount| (+ |$newcompErrorCount| 1))
  (setq erMsg
   (|processKeyedError|
    (|msgCreate| '|bug| |$nopos| erMsgKey erArgL "Bug!" optAttr)))
  (break)
  (|ncAbort|)))
```

———-

14.2.61 defun processMsgList

[erMsgSort p367]
[makeMsgFromLine p369]
[poGlobalLinePosn p72]
[getMsgPos p357]
[queueUpErrors p370]
[listOutputter p352]
[$noRepList p??]
[$outputList p??]

— **defun processMsgList** —

```
(defun |processMsgList| (erMsgList lineList)
 (let (|$noRepList| |$outputList| st globalNumOfLine msgLine)
 (declare (special |$noRepList| |$outputList|))
  (setq |$outputList| nil)
  (setq |$noRepList| nil)
  (setq erMsgList (|erMsgSort| erMsgList))
  (dolist (line lineList)
   (setq msgLine (|makeMsgFromLine| line))
   (setq |$outputList| (cons msgLine |$outputList|))
   (setq globalNumOfLine (|poGlobalLinePosn| (|getMsgPos| msgLine)))
   (setq erMsgList (|queueUpErrors| globalNumOfLine erMsgList)))
  (setq |$outputList| (append erMsgList |$outputList|))
  (setq st "---------SOURCE-TEXT-&-ERRORS-----------------------")
  (|listOutputter| (reverse |$outputList|))))
```

————-

14.2.62 defun erMsgSort

[erMsgSep p368]
[listSort p??]

— **defun erMsgSort** —

```
(defun |erMsgSort| (erMsgList)
 (let (msgWOPos msgWPos tmp)
  (setq tmp (|erMsgSep| erMsgList))
  (setq msgWPos (car tmp))
  (setq msgWOPos (cadr tmp))
  (setq msgWPos (|listSort| #'|erMsgCompare| msgWPos))
  (setq msgWOPos (reverse msgWOPos))
  (append msgWPos msgWOPos)))
```

———-

14.2.63 defun erMsgCompare

[compareposns p368]
[getMsgPos p357]

— **defun erMsgCompare** —

```
(defun |erMsgCompare| (ob1 ob2)
 (|compareposns| (|getMsgPos| ob2) (|getMsgPos| ob1)))
```

———-

14.2.64 defun compareposns

[poGlobalLinePosn p72]
[poCharPosn p375]

— **defun compareposns** —

```
(defun |compareposns| (a b)
 (let (c d)
  (setq c (|poGlobalLinePosn| a))
  (setq d (|poGlobalLinePosn| b))
  (if (equal c d)
   (not (< (|poCharPosn| a) (|poCharPosn| b)))
   (not (< c d)))))
```

———-

14.2.65 defun erMsgSep

[poNopos? p358]
[getMsgPos p357]

— **defun erMsgSep** —

```
(defun |erMsgSep| (erMsgList)
 (let (msgWOPos msgWPos)
  (dolist (msg erMsgList)
   (if (|poNopos?| (|getMsgPos| msg))
    (setq msgWOPos (cons msg msgWOPos))
```

```
    (setq msgWPos (cons msg msgWPos))))
  (list msgWPos msgWOPos)))
```

———-

14.2.66 defun makeMsgFromLine

[getLinePos p370]
[getLineText p370]
[poGlobalLinePosn p72]
[poLinePosn p359]
[strconc p**??**]
[rep p369]
[char p**??**]
[size p1039]
[$preLength p353]

— **defun makeMsgFromLine** —

```
(defun |makeMsgFromLine| (line)
 (let (localNumOfLine stNum globalNumOfLine textOfLine posOfLine)
 (declare (special |$preLength|))
  (setq posOfLine (|getLinePos| line))
  (setq textOfLine (|getLineText| line))
  (setq globalNumOfLine (|poGlobalLinePosn| posOfLine))
  (setq stNum (princ-to-string (|poLinePosn| posOfLine)))
  (setq localNumOfLine
    (strconc (|rep| #\space (- |$preLength| 7 (size stNum))) stNum))
  (list '|line| posOfLine nil nil (strconc "Line" localNumOfLine) textOfLine)))
```

———-

14.2.67 defun rep

TPDHERE: This function should be replaced by fillerspaces
— **defun rep 0** —

```
(defun |rep| (c n)
 (if (< 0 n)
  (make-string n :initial-element (character c))
  ""))
```

———-

14.2.68 defun getLinePos

— **defun getLinePos 0** —

```
(defun |getLinePos| (line) (car line))
```

———-

14.2.69 defun getLineText

— **defun getLineText 0** —

```
(defun |getLineText| (line) (cdr line))
```

———-

14.2.70 defun queueUpErrors

```
;queueUpErrors(globalNumOfLine,msgList)==
;    thisPosMsgs  := []
;    notThisLineMsgs := []
;    for msg in msgList _
;      while thisPosIsLess(getMsgPos msg,globalNumOfLine) repeat
;    --these are msgs that refer to positions from earlier compilations
;        if not redundant (msg,notThisPosMsgs) then
;           notThisPosMsgs := [msg,:notThisPosMsgs]
;        msgList := rest msgList
;    for msg in msgList _
;      while thisPosIsEqual(getMsgPos msg,globalNumOfLine) repeat
;       if not redundant (msg,thisPosMsgs) then
;           thisPosMsgs := [msg,:thisPosMsgs]
;       msgList := rest msgList
;    if thisPosMsgs then
;        thisPosMsgs := processChPosesForOneLine  thisPosMsgs
;        $outputList := NCONC(thisPosMsgs,$outputList)
;    if notThisPosMsgs then
;        $outputList := NCONC(notThisPosMsgs,$outputList)
;    msgList
```

[processChPosesForOneLine p374]
[$outputList p??]

— **defun queueUpErrors** —

```
(DEFUN |queueUpErrors| (|globalNumOfLine| |msgList|)
  (PROG (|notThisPosMsgs| |notThisLineMsgs| |thisPosMsgs|)
    (DECLARE (SPECIAL |$outputList|))
    (RETURN
      (PROGN
        (SETQ |thisPosMsgs| NIL)
        (SETQ |notThisLineMsgs| NIL)
        ((LAMBDA (|bfVar#7| |msg|)
           (LOOP
             (COND
               ((OR (ATOM |bfVar#7|)
                    (PROGN (SETQ |msg| (CAR |bfVar#7|)) NIL)
                    (NOT (|thisPosIsLess| (|getMsgPos| |msg|)
                             |globalNumOfLine|)))
                (RETURN NIL))
               ('T
                (PROGN
                  (COND
                    ((NULL (|redundant| |msg| |notThisPosMsgs|))
                     (SETQ |notThisPosMsgs|
                           (CONS |msg| |notThisPosMsgs|))))
                  (SETQ |msgList| (CDR |msgList|)))))
             (SETQ |bfVar#7| (CDR |bfVar#7|))))
         |msgList| NIL)
        ((LAMBDA (|bfVar#8| |msg|)
           (LOOP
             (COND
               ((OR (ATOM |bfVar#8|)
                    (PROGN (SETQ |msg| (CAR |bfVar#8|)) NIL)
                    (NOT (|thisPosIsEqual| (|getMsgPos| |msg|)
                             |globalNumOfLine|)))
                (RETURN NIL))
               ('T
                (PROGN
                  (COND
                    ((NULL (|redundant| |msg| |thisPosMsgs|))
                     (SETQ |thisPosMsgs| (CONS |msg| |thisPosMsgs|))))
                  (SETQ |msgList| (CDR |msgList|)))))
             (SETQ |bfVar#8| (CDR |bfVar#8|))))
         |msgList| NIL)
        (COND
          (|thisPosMsgs|
            (SETQ |thisPosMsgs|
                  (|processChPosesForOneLine| |thisPosMsgs|))
            (SETQ |$outputList| (NCONC |thisPosMsgs| |$outputList|))))
        (COND
          (|notThisPosMsgs|
            (SETQ |$outputList|
                  (NCONC |notThisPosMsgs| |$outputList|))))
        |msgList|))))
```

———-

14.2.71 defun thisPosIsLess

[poNopos? p358]
[poGlobalLinePosn p72]

— **defun thisPosIsLess** —

```
(defun |thisPosIsLess| (pos num)
 (unless (|poNopos?| pos) (< (|poGlobalLinePosn| pos) num)))
```

———-

14.2.72 defun thisPosIsEqual

[poNopos? p358]
[poGlobalLinePosn p72]

— **defun thisPosIsEqual** —

```
(defun |thisPosIsEqual| (pos num)
 (unless (|poNopos?| pos) (equal (|poGlobalLinePosn| pos) num)))
```

———-

14.2.73 defun redundant

```
redundant(msg,thisPosMsgs) ==
    found := NIL
    if msgNoRep? msg then
        for item in $noRepList repeat
            sameMsg?(msg,item) => return (found := true)
        $noRepList := [msg,$noRepList]
    found or MEMBER(msg,thisPosMsgs)
```

[msgNoRep? p373]
[sameMsg? p374]
[$noRepList p**??**]

— **defun redundant** —

```
(defun |redundant| (msg thisPosMsgs)
 (prog (found)
 (declare (special |$noRepList|))
  (return
   (progn
    (cond
     ((|msgNoRep?| msg)
      ((lambda (Var9 item)
       (loop
        (cond
         ((or (atom Var9) (progn (setq item (car Var9)) nil))
           (return nil))
         (t
          (cond
           ((|sameMsg?| msg item) (return (setq found t))))))
        (setq Var9 (cdr Var9))))
       |$noRepList| nil)
      (setq |$noRepList| (list msg |$noRepList|))))
   (or found (member msg thisPosMsgs))))))
```

———-

14.2.74 defvar $repGuys

— **initvars** —

```
(defvar |$repGuys| (list '|noRep| '|rep|))
```

———-

14.2.75 defun msgNoRep?

[getMsgCatAttr p356]

— **defun msgNoRep?** —

```
(defun |msgNoRep?| (msg) (eq (|getMsgCatAttr| msg '|$repGuys|) '|noRep|))
```

———-

14.2.76 defun sameMsg?

[getMsgKey p346]
[getMsgArgL p347]

— **defun sameMsg?** —

```
(defun |sameMsg?| (msg1 msg2)
 (and (equal (|getMsgKey| msg1) (|getMsgKey| msg2))
      (equal (|getMsgArgL| msg1) (|getMsgArgL| msg2))))
```

————-

14.2.77 defun processChPosesForOneLine

[posPointers p376]
[getMsgFTTag? p357]
[putFTText p377]
[poCharPosn p375]
[getMsgPos p357]
[getMsgPrefix p347]
[setMsgPrefix p347]
[strconc p**??**]
[size p1039]
[makeLeaderMsg p375]
[$preLength p353]

— **defun processChPosesForOneLine** —

```
(defun |processChPosesForOneLine| (msgList)
 (let (leaderMsg oldPre posLetter chPosList)
 (declare (special |$preLength|))
  (setq chPosList (|posPointers| msgList))
  (dolist (msg msgList)
   (when (|getMsgFTTag?| msg) (|putFTText| msg chPosList))
   (setq posLetter (cdr (assoc (|poCharPosn| (|getMsgPos| msg)) chPosList)))
   (setq oldPre (|getMsgPrefix| msg))
   (|setMsgPrefix| msg
    (strconc oldPre
     (make-string (- |$preLength| 4 (size oldPre))) posLetter)))
  (setq leaderMsg (|makeLeaderMsg| chPosList))
  (nconc msgList (list leaderMsg))))
```

————-

14.2.78 defun poCharPosn

— defun poCharPosn 0 —

```
(defun |poCharPosn| (posn)
 (cdr posn))
```

————-

14.2.79 defun makeLeaderMsg

```
makeLeaderMsg chPosList ==
    st := MAKE_-FULL_-CVEC ($preLength- 3)
    oldPos := -1
    for [posNum,:posLetter] in reverse chPosList repeat
        st := STRCONC(st, _
            rep(char ".", (posNum - oldPos - 1)),posLetter)
        oldPos := posNum
    ['leader,$nopos,'nokey,NIL,NIL,[st] ]
```

[$nopos p28]
[$preLength p353]

— defun makeLeaderMsg —

```
(defun |makeLeaderMsg| (chPosList)
 (let (posLetter posNum oldPos st)
 (declare (special |$nopos| |$preLength|))
  (setq st (make-string (- |$preLength| 3)))
  (setq oldPos -1)
  ((lambda (Var15 Var14)
   (loop
    (cond
     ((or (atom Var15) (progn (setq Var14 (car Var15)) nil))
       (return nil))
     (t
      (and (consp Var14)
           (progn
            (setq posNum (car Var14))
            (setq posLetter (cdr Var14))
            t)
           (progn
            (setq st
             (strconc st (|rep| (|char| '|.|) (- posNum oldPos 1)) posLetter))
            (setq oldPos posNum)))))
    (setq Var15 (cdr Var15))))
```

```
   (reverse chPosList) nil)
  (list '|leader| |$nopos| '|nokey| nil nil (list st))))
```

———-

14.2.80 defun posPointers

TPDHERE: getMsgFTTag is nonsense [poCharPosn p375]
[getMsgPos p357]
[IFCAR p**??**]
[getMsgPos2 p376]
[insertPos p377]
[getMsgFTTag p**??**]

— **defun posPointers** —

```
(defun |posPointers| (msgList)
 (let (posLetterList pos ftPosList posList increment pointers)
 (declare (special |getMsgFTTag|))
  (setq pointers "ABCDEFGHIJKLMONPQRS")
  (setq increment 0)
  (dolist (msg msgList)
   (setq pos (|poCharPosn| (|getMsgPos| msg)))
   (unless (equal pos (ifcar posList))
    (setq posList (cons pos posList)))
   ; this should probably read TPDHERE
   ; (when (eq (|getMsgPosTagOb| msg) 'fromto)
   (when (eq |getMsgFTTag| 'fromto)
    (setq ftPosList (cons (|poCharPosn| (|getMsgPos2| msg)) ftPosList))))
  (dolist (toPos ftPosList)
   (setq posList (|insertPos| toPos posList)))
  (dolist (pos posList)
   (setq posLetterList
    (cons (cons pos (elt pointers increment)) posLetterList))
   (setq increment (+ increment 1)))
  posLetterList)))
```

———-

14.2.81 defun getMsgPos2

[getMsgFTTag? p357]
[getMsgPosTagOb p346]
[ncBug p366]

— **defun getMsgPos2** —

```
(defun |getMsgPos2| (msg)
 (if (|getMsgFTTag?| msg)
  (caddr (|getMsgPosTagOb| msg))
  (|ncBug| "not a from to" nil)))
```

————-

14.2.82 defun insertPos

This function inserts a position in the proper place of a position list. This is used for the 2nd pos of a fromto [done p**??**]

— **defun insertPos 0** —

```
(defun |insertPos| (newPos posList)
 (let (pos top bot done)
 (setq bot (cons 0 posList))
 (do () (done)
  (setq top (cons (car bot) top))
  (setq bot (cdr bot))
  (setq pos (car bot))
  (setq done
   (cond
    ((< pos newPos) nil)
    ((equal pos newPos) t)
    ((< newPos pos)
     (setq top (cons newPos top))
     t))))
 (cons (cdr (reverse top)) bot)))
```

————-

14.2.83 defun putFTText

[getMsgFTTag? p357]
[poCharPosn p375]
[getMsgPos p357]
[setMsgText p347]
[getMsgText p347]
[getMsgPos2 p376]

— **defun putFTText** —

```
(defun |putFTText| (msg chPosList)
 (let (charMarker2 pos2 markingText charMarker pos tag)
  (setq tag (|getMsgFTTag?| msg))
  (setq pos (|poCharPosn| (|getMsgPos| msg)))
  (setq charMarker (cdr (assoc pos chPosList)))
  (cond
   ((eq tag 'from)
    (setq markingText (list "(from " charMarker " and on) "))
    (|setMsgText| msg (append markingText (|getMsgText| msg))))
   ((eq tag 'to)
    (setq markingText (list "(up to " charMarker ") "))
    (|setMsgText| msg (append markingText (|getMsgText| msg))))
   ((eq tag 'fromto)
    (setq pos2 (|poCharPosn| (|getMsgPos2| msg)))
    (setq charMarker2 (cdr (assoc pos2 chPosList)))
    (setq markingText (list "(from " charMarker " up to " charMarker2 ") "))
    (|setMsgText| msg (append markingText (|getMsgText| msg)))))))
```

———-

14.2.84 defun From

This is called from parameter list of nc message functions
— **defun From 0** —

```
(defun |From| (pos) (list 'from pos))
```

———-

14.2.85 defun To

This is called from parameter list of nc message functions
— **defun To 0** —

```
(defun |To| (pos) (list 'to pos))
```

———-

14.2.86 defun FromTo

This is called from parameter list of nc message functions
— **defun FromTo 0** —

```
(defun |FromTo| (pos1 pos2) (list 'fromto pos1 pos2))
```

Chapter 15

The Interpreter Syntax

15.1 syntax assignment

— assignment.help —

```
Immediate, Delayed, and Multiple Assignment

=======================================================================
Immediate Assignment
=======================================================================

A variable in Axiom refers to a value. A variable has a name beginning
with an uppercase or lowercase alphabetic character, "%", or "!".
Successive characters (if any) can be any of the above, digits, or "?".
Case is distinguished. The following are all examples of valid, distinct
variable names:

  a       tooBig?     a1B2c3%!?
  A       %j          numberOfPoints
  beta6   %J          numberofpoints

The ":=" operator is the immediate assignment operator. Use it to
associate a value with a variable. The syntax for immediate assignment
for a single variable is:

   variable := expression

The value returned by an immediate assignment is the value of expression.

  a := 1
    1
```

```
          Type: PositiveInteger
```

The right-hand side of the expression is evaluated, yielding 1. The value is then assigned to a.

```
  b := a
    1
          Type: PositiveInteger
```

The right-hand side of the expression is evaluated, yieldig 1. This value is then assigned to b. Thus a and b both have the value 1 after the sequence of assignments.

```
  a := 2
    2
          Type: PositiveInteger
```

What is the value of b if a is assigned the value 2?

```
  b
    1
          Type: PositiveInteger
```

The value of b is left unchanged.

This is what we mean when we say this kind of assignment is immediate. The variable b has no dependency on a after the initial assignment. This is the usual notion of assignment in programming languages such as C, Pascal, and Fortran.

```
========================================================================
Delayed Assignment
========================================================================
```

Axiom provides delayed assignment with "==". This implements a delayed evaluation of the right-hand side and dependency checking. The syntax for delayed assignment is

```
   variable == expression
```

The value returned by a delayed assignment is the unique value of Void.

```
  a == 1
          Type: Void

  b == a
          Type: Void
```

Using a and b as above, these are the corresponding delayed assignments.

```
  a
   Compiling body of rule a to compute value of type PositiveInteger
   1
           Type: PositiveInteger
```

The right-hand side of each delayed assignment is left unevaluated until the variables on the left-hand sides are evaluated.

```
  b
   Compiling body of rule b to compute value of type PositiveInteger
   1
           Type: PositiveInteger
```

This gives the same results as before. But if we change a to 2

```
  a == 2
   Compiled code for a has been cleared.
   Compiled code for b has been cleared.
   1 old definition(s) deleted for function or rule a
           Type: Void
```

Then a evaluates to 2, as expected

```
  a
   Compiling body of rule a to compute value of type PositiveInteger
   2
           Type: PositiveInteger
```

but the value of b reflects the change to a

```
  b
   Compiling body of rule b to compute value of type PositiveInteger
   2
           Type: PositiveInteger
```

```
========================================================================
Multiple Immediate Assignments
========================================================================
```

It is possible to set several variables at the same time by using a tuple of variables and a tuple of expressions. A tuple is a collection of things separated by commas, often surrounded by parentheses. The syntax for multiple immediate assignment is

```
 ( var1, var2, ..., varN ) := ( expr1, expr2, ..., exprN )
```

The value returned by an immediate assignment is the value of exprN.

```
 ( x, y ) := ( 1, 2 )
   2
```

```
          Type: PositiveInteger

This sets x to 1 and y to 2. Multiple immediate assignments are parallel
in the sense that the expressions on the right are all evaluated before
any assignments on the left are made. However, the order of evaluation
of these expressions is undefined.

 ( x, y ) := ( y, x )
   1
          Type: PositiveInteger

  x
   2
          Type: PositiveInteger

The variable x now has the previous value of y.

  y
   1
          Type: PositiveInteger

The variable y now has the previous value of x.

There is no syntactic form for multiple delayed assignments.
```

15.2 syntax blocks

— blocks.help —

```
=======================================================================
Blocks
=======================================================================

A block is a sequence of expressions evaluated in the order that they
appear, except as modified by control expressions such as leave, return,
iterate, and if-then-else constructions. The value of a block is the
value of the expression last evaluated in the block.

To leave a block early, use "=>". For example,

    i < 0 => x

The expression before the "=>" must evaluate to true or false. The
expression following the "=>" is the return value of the block.
```

```
A block can be constructed in two ways:

  1. the expressions can be separated by semicolons and the resulting
     expression surrounded by parentheses, and
  2. the expressions can be written on succeeding lines with each line
     indented the same number of spaces (which must be greater than zero).
     A block entered in this form is called a pile

Only the first form is available if you are entering expressions directly
to Axiom. Both forms are available in .input files. The syntax for a simple
block of expressions entered interactively is

  ( expression1 ; expression2 ; ... ; expressionN )

The value returned by a block is the value of an "=>" expression, or
expressionN if no "=>" is encountered.

In .input files, blocks can also be written in piles. The examples
given here are assumed to come from .input files.

  a :=
    i := gcd(234,672)
    i := 2*i**5 - i + 1
    1 / i

      1
    -----
    23323
              Type: Fraction Integer

In this example, we assign a rational number to a using a block consisting
of three expressions. This block is written as a pile. Each expression in
the pile has the same indentation, in this case two spaces to the right of
the first line.

  a := ( i := gcd(234,672); i := 2*i**5 - i + 1; 1 / i )

      1
    -----
    23323
              Type: Fraction Integer

Here is the same block written on one line. This is how you are required
to enter it at the input prompt.

  ( a := 1; b := 2; c := 3; [a,b,c] )
    [1,2,3]
              Type: List PositiveInteger
```

```
AAxiom gives you two ways of writing a block and the preferred way in
an .input file is to use a pile. Roughly speaking, a pile is a block
whose consituent expressions are indented the same amount. You begin a
pile by starting a new line for the first expression, indenting it to
the right of the previous line. You then enter the second expression on
a new line, vertically aligning it with the first line. And so on. If
you need to enter an inner pile, further indent its lines to the right
of the outer pile. Axiom knows where a pile ends. It ends when a subsequent
line is indented to the left of the pile or the end of the file.

Also See:
o )help if
o )help repeat
o )help while
o )help for
o )help suchthat
o )help parallel
o )help lists
```

———-

[1]

15.3 system clef

— clef.help —

```
Entering printable keys generally inserts new text into the buffer (unless
in overwrite mode, see below).  Other special keys can be used to modify
the text in the buffer.  In the description of the keys below, ^n means
Control-n, or holding the CONTROL key down while pressing "n".  Errors
will ring the terminal bell.

^A/^E   : Move cursor to beginning/end of the line.
^F/^B   : Move cursor forward/backward one character.
^D      : Delete the character under the cursor.
^H, DEL : Delete the character to the left of the cursor.
^K      : Kill from the cursor to the end of line.
^L      : Redraw current line.
^O      : Toggle overwrite/insert mode. Initially in insert mode. Text
          added in overwrite mode (including yanks) overwrite
          existing text, while insert mode does not overwrite.
^P/^N   : Move to previous/next item on history list.
```

[1] “if” (15.6 p 393) “repeat” (15.10 p 400) “while” (67.1.2 p 1033) “for” (15.5 p 389) “suchthat” (15.11 p 404) “parallel” (15.9 p 397) “lists” (?? p ??)

```
^R/^S   : Perform incremental reverse/forward search for string on
          the history list.  Typing normal characters adds to the current
          search string and searches for a match. Typing ^R/^S marks
          the start of a new search, and moves on to the next match.
          Typing ^H or DEL deletes the last character from the search
          string, and searches from the starting location of the last search.
          Therefore, repeated DEL's appear to unwind to the match nearest
          the point at which the last ^R or ^S was typed.  If DEL is
          repeated until the search string is empty the search location
          begins from the start of the history list.  Typing ESC or
          any other editing character accepts the current match and
          loads it into the buffer, terminating the search.
^T      : Toggle the characters under and to the left of the cursor.
^Y      : Yank previously killed text back at current location.  Note that
          this will overwrite or insert, depending on the current mode.
^U      : Show help (this text).
TAB     : Perform command completion based on word to the left of the cursor.
          Words are deemed to contain only the alphanumeric and the % ! ? _
          characters.
NL, CR  : returns current buffer to the program.

DOS and ANSI terminal arrow key sequences are recognized, and act like:

  up    : same as ^P
  down  : same as ^N
  left  : same as ^B
  right : same as ^F
```

———-

15.4 syntax collection

— collection.help —

```
========================================================================
Collection -- Creating Lists and Streams with Iterators
========================================================================

All of the loop expressions which do not use the repeat leave or
iterate words can be used to create lists and streams. For example:

This creates a simple list of the integers from 1 to 10:

  list := [i for i in 1..10]
   [1,2,3,4,5,6,7,8,9,10]
                     Type: List PositiveInteger
```

```
Create a stream of the integers greater than or equal to 1:

  stream := [i for i in 1..]
   [1,2,3,4,5,6,7,...]
                      Type: Stream PositiveInteger

This is a list of the prime numbers between 1 and 10, inclusive:

  [i for i in 1..10 | prime? i]
   [2,3,5,7]
                      Type: List PositiveInteger

This is a stream of the prime integers greater than or equal to 1:

  [i for i in 1.. | prime? i]
   [2,3,5,7,11,13,17,...]
                      Type: Stream PositiveInteger

This is a list of the integers between 1 and 10, inclusive, whose
squares are less than 700:

  [i for i in 1..10 while i*i < 700]
   [1,2,3,4,5,6,7,8,9,10]
                      Type: List PositiveInteger

This is a stream of the integers greater than or equal to 1 whose
squares are less than 700:

  [i for i in 1.. while i*i < 700]
   [1,2,3,4,5,6,7,...]
                      Type: Stream PositiveInteger

The general syntax of a collection is

  [ collectExpression iterator1 iterator2 ... iteratorN ]

where each iterator is either a for or a while clause. The loop
terminates immedidately when the end test of any iterator succeeds
or when a return expression is evaluated in collectExpression. The
value returned by the collection is either a list or a stream of
elements, one for each iteration of the collectExpression.

Be careful when you use while to create a stream. By default Axiom
tries to compute and display the first ten elements of a stream. If
the while condition is not satisfied quickly, Axiom can spend a long
(potentially infinite) time trying to compute the elements. Use

  )set streams calculate
```

```
to change the defaults to something else. This also affects the number
of terms computed and displayed for power series. For the purposes of
these examples we have use this system command to display fewer than
ten terms.
```

———-

15.5 syntax for

— for.help —

```
========================================================================
for loops
========================================================================

Axiom provide the for and in keywords in repeat loops, allowing you
to integrate across all elements of a list, or to have a variable take
on integral values from a lower bound to an upper bound. We shall refer
to these modifying clauses of repeat loops as for clauses. These clauses
can be present in addition to while clauses (See )help while). As with
all other types of repeat loops, leave (see )help leave) can be used to
prematurely terminate evaluation of the loop.

The syntax for a simple loop using for is

  for iterator repeat loopbody

The iterator has several forms. Each form has an end test which is
evaluted before loopbody is evaluated. A for loop terminates immediately
when the end test succeeds (evaluates to true) or when a leave or return
expression is evaluated in loopbody. The value returned by the loop is
the unique value of Void.

========================================================================
for i in n..m repeat
========================================================================

If for is followed by a variable name, the in keyword and then an integer
segment of the form n..m, the end test for this loop is the predicate
i > m. The body of the loop is evaluated m-n+1 times if this number is
greater than 0. If this number is less than or equal to 0, the loop body
is not evaluated at all.

The variable i has the value n, n+1, ..., m for successive iterations
of the loop body. The loop variable is a local variable within the loop
body. Its value is not available outside the loop body and its value and
```

```
type within the loop body completely mask any outer definition of a
variable with the same name.

  for i in 10..12 repeat output(i**3)
   1000
   1331
   1728
                      Type: Void

The loop prints the values of 10^3, 11^3, and 12^3.

  a := [1,2,3]
   [1,2,3]
                      Type: List PositiveInteger

  for i in 1..#a repeat output(a.i)
   1
   2
   3
                      Type: Void

Iterate across this list using "." to access the elements of a list
and the # operation to count its elements.

This type of iteration is applicable to anything that uses ".". You
can also use it with functions that use indices to extract elements.

   m := matrix [ [1,2],[4,3],[9,0] ]
    +-    -+
    | 1  2 |
    | 4  3 |
    | 9  0 |
    +-    -+
                      Type: Matrix Integer

Define m to be a matrix.

   for i in 1..nrows(m) repeat output row(m.i)
    [1,2]
    [4,3]
    [9,0]
                      Type: Void

Display the rows of m.

You can iterate with for-loops.

   for i in 1..5 repeat
     if odd?(i) then iterate
     output(i)
```

```
    2
    4
                        Type: Void

Display the even integers in a segment.

========================================================================
for i in n..m by s repeat
========================================================================

By default, the difference between values taken on by a variable in
loops such as

  for i in n..m repeat ...

is 1. It is possible to supply another, possibly negative, step value
by using the by keyword along with for and in. Like the upper and lower
bounds, the step value following the by keyword must be an integer. Note
that the loop

  for i in 1..2 by 0 repeat output(i)

will not terminate by itself, as the step value does not change the
index from its initial value of 1.

  for i in 1..5 by 2 repeat output(i)
   1
   3
   5
                        Type: Void

This expression displays the odd integers between two bounds.

  for i in 5..1 by -2 repeat output(i)
   5
   3
   1
                        Type: Void

Use this to display the numbers in reverse order.

========================================================================
for i in n.. repeat
========================================================================

If the value after the ".." is omitted, the loop has no end test. A
potentially infinite loop is thus created. The variable is given the
successive values n, n+1, n+2, ... and the loop is terminated only
if a leave or return expression is evaluated in the loop body. However,
you may also add some other modifying clause on the repeat, for example,
```

```
a while clause, to stop the loop.

  for i in 15.. while not prime?(i) repeat output(i)
   15
   16
                      Type: Void

This loop displays the integers greater than or equal to 15 and less
than the first prime number greater than 15.

========================================================================
for x in l repeat
========================================================================

Another variant of the for loop has the form:

  for x in list repeat loopbody

This form is used when you want to iterate directly over the elements
of a list. In this form of the for loop, the variable x takes on the
value of each successive element in l. The end test is most simply
stated in English: "are there no more x in l?"

  l := [0, -5, 3]
   [0, -5, 3]
                      Type: List Integer

  for x in l repeat output(x)
   0
   -5
   3
                      Type: Void

This displays all of the elements of the list l, one per line.

Since the list constructing expression

  expand [n..m]

creates the list

  [n, n+1, ..., m]

you might be tempted to think that the loops

  for i in n..m repeat output(i)

and

  for x in expand [n..m] repeat output(x)
```

```
are equivalent. The second form first creates the expanded list
(no matter how large it might be) and then does the iteration. The
first form potentially runs in much less space, as the index variable
i is simply incremented once per loop and the list is not actually
created. Using the first form is much more efficient.

Of course, sometimes you really want to iterate across a specific list.
This displays each of the factors of 2400000:

  for f in factors(factor(2400000)) repeat output(f)
   [factor= 2, exponent= 8]
   [factor= 3, exponent= 1]
   [factor= 5, exponent= 5]
                      Type: Void
```

———-

15.6 syntax if

— if.help —

```
=====================================================================
If-then-else
=====================================================================

Like many other programming languages, Axiom uses the three keywords
if, then, and else to form conditional expressions. The else part of
the conditional is optional. The expression between the if and then
keywords is a predicate: an expression that evaluates to or is
convertible to either true or false, that is, a Boolean.

The syntax for conditional expressions is

   if predicate then expression1 else expression2

where the "else expression2" part is optional. The value returned from
a conditional expression is expression1 if the predicate evaluates to
true and expression2 otherwise. If no else clause is given, the value
is always the unique value of Void.

An if-then-else expression always returns a value. If the else clause
is missing then the entire expression returns the unique value of Void.
If both clauses are present, the type of the value returned by if is
obtained by resolving the types of the values of the two clauses.
```

```
The predicate must evaluate to, or be convertible to, an object of type
Boolean: true or false. By default, the equal sign "=" creates an equation.

   x + 1 = y
    x + 1 = y
                Type: Equation Polynomial Integer

This is an equation, not a boolean condition. In particular, it is
an object of type Equation Polynomial Integer.

However, for predicates in if expressions, Axiom places a default
target type of Boolean on the predicate and equality testing is performed.
Thus you need not qualify the "=" in any way. In other contexts you may
need to tell Axiom that you want to test for equality rather than create
an equation. In these cases, use "@" and a target type of Boolean.

The compound symbol meaning "not equal" in Axiom is "~=". This can be
used directly without a package call or a target specification. The
expression "a ~= b" is directly translated to "not(a = b)".

Many other functions have return values of type Boolean. These include
<, <=, >, >=, ~=, and member?. By convention, operations with names
ending in "?" return Boolean values.

The usual rules for piles are suspended for conditional expressions. In
.input files, the then and else keywords can begin in the same column
as the corresponding if by may also appear to the right. Each of the
following styles of writing if-then-else expressions is acceptable:

  if i>0 then output("positive") else output("nonpositive")

  if i>0 then output("positive")
    else output("nonpositive")

  if i>0 then output("positive")
  else output("nonpositive")

  if i>0
  then output("positive")
  else output("nonpositive")

  if i>0
    then output("positive")
    else output("nonpositive")

A block can follow the then or else keywords. In the following two
assignments to a, the then and else clauses each are followed by two
line piles. The value returned in each is the value of the second line.

  a :=
```

```
    if i > 0 then
      j := sin(i * pi())
      exp(j + 1/j)
    else
      j := cos(i * 0.5 * pi())
      log(abs(j)**5 + i)


  a :=
    if i > 0
      then
        j := sin(i * pi())
        exp(j + 1/j)
      else
        j := cos(i * 0.5 * pi())
        log(abs(j)**5 + i)
```

These are both equivalent to the following:

```
  a :=
    if i > 0 then (j := sin(i * pi()); exp(j + 1/j))
    else (j := cos(i * 0.5 * pi()); log(abs(j)**5 + i))
```

———-

15.7 syntax iterate

— **iterate.help** —

```
====================================================================
iterate in loops
====================================================================

Axiom provides an iterate expression that skips over the remainder
of a loop body and starts the next loop execution. We first initialize
a counter.

  i := 0
   0
                      Type: NonNegativeInteger

Display the even integers from 2 to 5:

  repeat
    i := i + 1
    if i > 5 then leave
```

```
    if odd?(i) then iterate
    output(i)
   2
   4
                        Type: Void
```

———-

15.8 syntax leave

— **leave.help** —

```
========================================================================
leave in loops
========================================================================

The leave keyword is often more useful in terminating a loop. A
leave causes control to transfer to the expression immediately following
the loop. As loops always return the unique value of Void, you cannot
return a value with leave. That is, leave takes no argument.

  f() ==
    i := 1
    repeat
      if factorial(i) > 1000 then leave
      i := i + 1
    i
                        Type: Void

This example is a modification of the last example in the previous
section. Instead of using return we'll use leave.

  f()
   7
                        Type: PositiveInteger

The loop terminates when factorial(i) gets big enough. The last line
of the function evaluates to the corresponding "good" value of i
and the function terminates, returning that value.

You can only use leave to terminate the evaluation of one loop. Lets
consider a loop within a loop, that is, a loop with a nested loop.
First, we initialize two counter variables.

  (i,j) := (1,1)
   1
```

```
                    Type: PositiveInteger

  repeat
    repeat
      if (i + j) > 10 then leave
      j := j + 1
    if (i + j) > 10 then leave
    i := i + 1
                    Type: Void
```

Nested loops must have multiple leave expressions at the appropriate nesting level. How would you rewrite this so (i + j) > 10 is only evaluated once?

```
========================================================================
leave vs => in loop bodies
========================================================================
```

Compare the following two loops:

```
  i := 1                      i := 1
  repeat                      repeat
    i := i + 1                  i := i + 1
    i > 3 => i                  if i > 3 then leave
    output(i)                   output(i)
```

In the example on the left, the values 2 and 3 for i are displayed but then the "=>" does not allow control to reach the call to output again. The loop will not terminate until you run out of space or interrupt the execution. The variable i will continue to be incremented because the "=>" only means to leave the block, not the loop.

In the example on the right, upon reaching 4, the leave will be executed, and both the block and the loop will terminate. This is one of the reasons why both "=>" and leave are provided. Using a while clase with the "=>" lets you simulate the action of leave.

———-

15.9 syntax parallel

— **parallel.help** —

```
========================================================================
parallel iteration
========================================================================
```

```
Sometimes you want to iterate across two lists in parallel, or perhaps
you want to traverse a list while incrementing a variable.

The general syntax of a repeat loop is

 iterator1, iterator2, ..., iteratorN repeat loopbody

where each iterator is either a for or a while clause. The loop
terminates immediately when the end test of any iterator succeeds or
when a leave or return expression is evaluated in loopbody. The value
returned by the loop is the unique value of Void.

  l := [1,3,5,7]
   [1,3,5,7]
                      Type: List PositiveInteger

  m := [100,200]
   [100,200]
                      Type: List PositiveInteger

  sum := 0
   0
                      Type: NonNegativeInteger

Here we write a loop to iterate across two lists, computing the sum
of the pairwise product of the elements:

  for x in l for y in m repeat
    sum := sum + x*y
                      Type: Void

The last two elements of l are not used in the calculation because
m has two fewer elements than l.

  sum
   700
                      Type: NonNegativeInteger

This is the "dot product".

Next we write a loop to compute the sum of the products of the loop
elements with their positions in the loop.

  l := [2,3,5,7,11,13,17,19,23,29,31,37]
   [2,3,5,7,11,13,17,19,23,29,31,37]
                      Type: List PositiveInteger

  sum := 0
   0
```

```
                    Type: NonNegativeInteger

  for i in 0.. for x in l repeat sum := i * x
                    Type: Void
```

Here looping stops when the list l is exhaused, even though the
for i in 0.. specifies no terminating condition.

```
  sum
   407
                    Type: NonNegativeInteger
```

When "|" is used to qualify any of the for clauses in a parallel
iteration, the variables in the predicates can be from an outer
scope or from a for clause in or to the left of the modified clause.

This is correct:

```
  for i in 1..10 repeat
    for j in 200..300 | ood? (i+j) repeat
      output [i,j]
```

But this is not correct. The variable j has not been defined outside
the inner loop:

```
  for i in 1..01 | odd? (i+j) repeat -- wrong, j not defined
    for j in 200..300 repeat
      output [i,j]
```

It is possible to mix several of repeat modifying clauses on a loop:

```
  for i in 1..10
    for j in 151..160 | odd? j
      while i + j < 160 repeat
        output [i,j]
   [1,151]
   [3,153]
                    Type: Void
```

Here are useful rules for composing loop expressions:

1. while predicates can only refer to variables that are global (or
   in an outer scope) or that are defined in for clauses to the left
   of the predicate.
2. A "such that" predicate (somthing following "|") must directly
   follow a for clause and can only refer to variables that are
   global (or in an outer scope) or defined in the modified for clause
   or any for clause to the left.

———-

15.10 syntax repeat

— **repeat.help** —

```
=======================================================================
Repeat Loops
=======================================================================

A loop is an expression that contains another expression, called the loop
body, which is to be evaluated zero or more times. All loops contain the
repeat keyword and return the unique value of Void. Loops can contain
inner loops to any depth.

The most basic loop is of the form

  repeat loopbody

Unless loopbody contains a leave or return expression, the loop repeats
foreer. The value returned by the loop is the unique value of Void.

Axiom tries to determine completely the type of every object in a loop
and then to translate the loop body to Lisp or even to machine code. This
translation is called compilation.

If Axiom decides that it cannot compile the loop, it issues a message
stating the problem and then the following message:

  We will attemp to step through and interpret the code

It is still possible that Axiom can evalute the loop but in interpret-code
mode.

=======================================================================
Return in Loops
=======================================================================

A return expression is used to exit a function with a particular value.
In particular, if a return is in a loop within the function, the loop
is terminated whenever the return is evaluated.

  f() ==
    i := 1
    repeat
      if factorial(i) > 1000 then return i
      i := i + 1
```

```
                    Type: Void

  f()
                    Type: Void
```

When factorial(i) is big enough, control passes from inside the loop all the way outside the function, returning the value of i (so we think). What went wrong? Isn't it obvious that this function should return an integer? Well, Axiom makes no attempt to analyze the structure of a loop to determine if it always returns a value because, in general, this is impossible. So Axiom has this simple rule: the type of the function is determined by the type of its body, in this case a block. The normal value of a block is the value of its last expression, in this case, a loop. And the value of every loop is the unique value of Void. So the return type of f is Void.

There are two ways to fix this. The best way is for you to tell Axiom what the return type of f is. You do this by giving f a declaration

```
   f:() -> Integer
```

prior to calling for its value. This tells Axiom "trust me -- an integer is returned". Another way is to add a dummy expression as follows.

```
  f() ==
    i := 1
    repeat
      if factorial(i) > 1000 then return i
      i := i + 1
    0
                    Type: Void
```

Note that the dummy expression will never be evaluated but it is the last expression in the function and will determine the return type.

```
  f()
   7
                    Type: PositiveInteger
```

```
========================================================================
leave in loops
========================================================================
```

The leave keyword is often more useful in terminating a loop. A leave causes control to transfer to the expression immediately following the loop. As loops always return the unique value of Void, you cannot return a value with leave. That is, leave takes no argument.

```
  f() ==
    i := 1
```

```
    repeat
      if factorial(i) > 1000 then leave
      i := i + 1
    i
                        Type: Void
```

This example is a modification of the last example in the previous section. Instead of using return we'll use leave.

```
  f()
   7
                        Type: PositiveInteger
```

The loop terminates when factorial(i) gets big enough. The last line of the function evaluates to the corresponding "good" value of i and the function terminates, returning that value.

You can only use leave to terminate the evaluation of one loop. Lets consider a loop within a loop, that is, a loop with a nested loop. First, we initialize two counter variables.

```
  (i,j) := (1,1)
   1
                        Type: PositiveInteger

  repeat
    repeat
      if (i + j) > 10 then leave
      j := j + 1
    if (i + j) > 10 then leave
    i := i + 1
                        Type: Void
```

Nested loops must have multiple leave expressions at the appropriate nesting level. How would you rewrite this so (i + j) > 10 is only evaluated once?

```
========================================================================
leave vs => in loop bodies
========================================================================
```

Compare the following two loops:

```
  i := 1                        i := 1
  repeat                        repeat
    i := i + 1                    i := i + 1
    i > 3 => i                    if i > 3 then leave
    output(i)                     output(i)
```

In the example on the left, the values 2 and 3 for i are displayed but

```
then the "=>" does not allow control to reach the call to output again.
The loop will not terminate until you run out of space or interrupt the
execution. The variable i will continue to be incremented because the
"=>" only means to leave the block, not the loop.

In the example on the right, upon reaching 4, the leave will be executed,
and both the block and the loop will terminate. This is one of the reasons
why both "=>" and leave are provided. Using a while clase with the "=>"
lets you simulate the action of leave.

========================================================================
iterate in loops
========================================================================

Axiom provides an iterate expression that skips over the remainder
of a loop body and starts the next loop execution. We first initialize
a counter.

  i := 0
   0
                      Type: NonNegativeInteger

Display the even integers from 2 to 5:

  repeat
    i := i + 1
    if i > 5 then leave
    if odd?(i) then iterate
    output(i)
   2
   4
                      Type: Void

Also See:
o )help blocks
o )help if
o )help while
o )help for
o )help suchthat
o )help parallel
o )help lists
```

2

[2] "blocks" (15.2 p 384) "if" (15.6 p 393) "while" (67.1.2 p 1033) "for" (15.5 p 389) "suchthat" (15.11 p 404) "parallel" (15.9 p 397) "lists" (**??** p **??**)

15.11 syntax suchthat

— **suchthat.help** —

```
========================================================================
Such that predicates
========================================================================

A for loop can be followed by a "|" and then a predicate. The predicate
qualifies the use of the values from the iterator that follows the for.
Think of the vertical bar "|" as the phrase "such that".

  for n in 0..4 | odd? n repeat output n
   1
   3
                      Type: Void

This loop expression prints out the integers n in the given segment
such that n is odd.

A for loop can also be written

  for iterator | predicate repeat loopbody

which is equivalent to:

  for iterator repeat if predicate then loopbody else iterate

The predicate need not refer only to the variable in the for clause.
Any variable in an outer scope can be part of the predicate.

  for i in 1..50 repeat
    for j in 1..50 | factorial(i+j) < 25 repeat
      output [i,j]
   [1,1]
   [1,2]
   [1,3]
   [2,1]
   [2,2]
   [3,1]
                      Type: Void
```

———-

15.12 syntax syntax

— **syntax.help** —

```
The Axiom Interactive Language has the following features documented here.

More information is available by typing

  )help feature

where feature is one of:

  assignment -- Immediate and delayed assignments
  blocks     -- Blocks of expressions
  collection -- creating lists with iterators
  for        -- for loops
  if         -- If-then-else statements
  iterate    -- using iterate in loops
  leave      -- using leave in loops
  parallel   -- parallel iterations
  repeat     -- repeat loops
  suchthat   -- suchthat predicates
  while      -- while loops
```

———-

15.13 syntax while

— **while.help** —

```
========================================================================
while loops
========================================================================

The repeat in a loop can be modified by adding one or more while
clauses. Each clause contains a predicate immediately following the
while keyword. The predicate is tested before the evaluation of the
body of the loop. The loop body is evaluated whenever the predicate
in a while clause is true.

The syntax for a simple loop using while is

  while predicate repeat loopbody
```

```
The predicate is evaluated before loopbody is evaluated. A while loop
terminates immediately when predicate evaluates to false or when a
leave or return expression is evaluted. See )help repeat for more
information on leave and return.

Here is a simple example of using while in a loop. We first initialize
the counter.

  i := 1
   1
                      Type: PositiveInteger

  while i < 1 repeat
    output "hello"
    i := i + 1
                      Type: Void

The steps involved in computing this example are
 (1) set i to 1
 (2) test the condition i < 1 and determine that it is not true
 (3) do not evaluate the loop body and therefore do not display "hello"

  (x, y) := (1, 1)
   1
                      Type: PositiveInteger

If you have multiple predicates to be tested use the logical and
operation to separate them. Axiom evaluates these predicates from
left to right.

  while x < 4 and y < 10 repeat
    output [x,y]
    x := x + 1
    y := y + 2
   [1,1]
   [2,3]
   [3,5]
                      Type: Void


A leave expression can be included in a loop body to terminate a loop
even if the predicate in any while clauses are not false.

  (x, y) := (1, 1)
   1
                      Type: PositiveInteger

  while x < 4 and y < 10 repeat
    if x + y > 7 then leave
```

```
 output [x,y]
 x := x + 1
 y := y + 2
[1,1]
[2,3]
                    Type: Void
```

———————-

Chapter 16

Abstract Syntax Trees (ptrees)

```
Abstract Syntax Trees

These functions create and examine abstract
syntax trees.  These are called pform, for short.

!! This file also contains constructors for concrete syntax, although
!! they should be somewhere else.

THE PFORM DATA STRUCTURE
 Leaves: [hd, tok, pos]
 Trees:  [hd, tree, tree, ...]
 hd is either an id or (id . alist)
```

16.0.1 defun Construct a leaf token

The tokConstruct function is a constructer and selectors for leaf tokens. A leaf token looks like [head, token, position] where head is either an id or (id . alist) [ifcar p**??**]
[pfNoPosition? p410]
[ncPutQ p414]

— defun tokConstruct —

```
(defun |tokConstruct| (head token &rest position)
 (let (result)
  (setq result (cons head token))
  (cond
   ((ifcar position)
    (cond
     ((|pfNoPosition?| (car position)) result)
     (t (|ncPutQ| result '|posn| (car position)) result)))
```

```
   (t result))))
```

———-

16.0.2 defun Return a part of a node

[ifcar p??]

— **defun pfAbSynOp** —

```
(defun |pfAbSynOp| (form)
 (let (hd)
  (setq hd (car form))
  (or (ifcar hd) hd)))
```

———-

16.0.3 defun Compare a part of a node

[eqcar p??]

— **defun pfAbSynOp?** —

```
(defun |pfAbSynOp?| (form op)
 (let (hd)
  (setq hd (car form))
  (or (eq hd op) (eqcar hd op))))
```

———-

16.0.4 defun pfNoPosition?

[poNoPosition? p411]

— **defun pfNoPosition?** —

```
(defun |pfNoPosition?| (pos)
 (|poNoPosition?| pos))
```

———-

16.0.5 defun poNoPosition?

[eqcar p??]

— **defun poNoPosition? 0** —

```
(defun |poNoPosition?| (pos)
 (eqcar pos '|noposition|))
```

———-

16.0.6 defun tokType

[ncTag p413]

— **defun tokType** —

```
(defun |tokType| (x) (|ncTag| x))
```

———-

16.0.7 defun tokPart

— **defun tokPart 0** —

```
(defun |tokPart| (x) (cdr x))
```

———-

16.0.8 defun tokPosn

[qassq p??]
[ncAlist p413]
[pfNoPosition p412]

— **defun tokPosn** —

```
(defun |tokPosn| (x)
 (let (a)
  (setq a (qassq '|posn| (|ncAlist| x)))
  (cond
```

```
   (a (cdr a))
   (t (|pfNoPosition|))))
```

———-

16.0.9 defun pfNoPosition

[poNoPosition p412]

— **defun pfNoPosition** —

```
(defun |pfNoPosition| () (|poNoPosition|))
```

———-

16.0.10 defun poNoPosition

[$nopos p28]

— **defun poNoPosition 0** —

```
(defun |poNoPosition| ()
 (declare (special |$nopos|))
 |$nopos|)
```

———-

Chapter 17

Attributed Structures

For objects which are pairs where the CAR field is either just a tag (an identifier) or a pair which is the tag and an association list.

17.0.11 defun ncTag

Pick off the tag [ncBug p366]
[qcar p**??**]
[identp p1040]

— **defun ncTag** —

```
(defun |ncTag| (x)
 (cond
  ((null (consp x)) (|ncBug| 's2cb0031 nil))
  (t
   (setq x (qcar x))
   (cond
    ((identp x) x)
    ((null (consp x)) (|ncBug| 's2cb0031 nil))
    (t (qcar x))))))
```

———-

17.0.12 defun ncAlist

Pick off the property list [ncBug p366]
[qcar p**??**]
[identp p1040]
[qcdr p**??**]

— **defun ncAlist** —

```
(defun |ncAlist| (x)
 (cond
  ((null (consp x)) (|ncBug| 's2cb0031 nil))
  (t
   (setq x (qcar x))
   (cond
    ((identp x) nil)
    ((null (consp x)) (|ncBug| 's2cb0031 nil))
    (t (qcdr x))))))
```

———-

17.0.13 defun ncEltQ

Get the entry for key k on x's association list [qassq p**??**]
[ncAlist p413]
[ncBug p366]

— **defun ncEltQ** —

```
(defun |ncEltQ| (x k)
 (let (r)
  (setq r (qassq k (|ncAlist| x)))
  (cond
   ((null r) (|ncBug| 's2cb0007 (list k)))
   (t (cdr r)))))
```

———-

17.0.14 defun ncPutQ

```
;-- Put (k . v) on the association list of x and return v
;-- case1: ncPutQ(x,k,v) where k is a key (an identifier), v a value
;--        put the pair (k . v) on the association list of x and return v
;-- case2: ncPutQ(x,k,v) where k is a list of keys, v a list of values
;--        equivalent to [ncPutQ(x,key,val) for key in k for val in v]
;ncPutQ(x,k,v) ==
;   LISTP k =>
;      for key in k for val in v repeat ncPutQ(x,key,val)
;      v
;   r := QASSQ(k,ncAlist x)
;   if NULL r then
```

```
;      r := CONS( CONS(k,v), ncAlist x)
;      RPLACA(x,CONS(ncTag x,r))
;   else
;      RPLACD(r,v)
;   v
```

[qassq p??]
[ncAlist p413]
[ncTag p413]

— defun ncPutQ —

```
(defun |ncPutQ| (x k v)
 (let (r)
  (cond
   ((listp k)
    ((lambda (Var1 key Var2 val)
     (loop
      (cond
       ((or (atom Var1)
            (progn (setq key (car Var1)) nil)
            (atom Var2)
            (progn (setq val (car Var2)) nil))
          (return nil))
        (t
          (|ncPutQ| x key val)))
      (setq Var1 (cdr Var1))
      (setq Var2 (cdr Var2))))
     k nil v nil)
    v)
   (t
    (setq r (qassq k (|ncAlist| x)))
    (cond
     ((null r)
      (setq r (cons (cons k v) (|ncAlist| x)))
      (rplaca x (cons (|ncTag| x) r)))
     (t
      (rplacd r v)))
    v))))
```

———-

Chapter 18

Function Selection

```
New Selection of Modemaps

selection of applicable modemaps is done in two steps:
  first it tries to find a modemap inside an argument domain, and if
  this fails, by evaluation of pattern modemaps
the result is a list of functions with signatures, which have the
  following form:
  [sig,elt,cond] where
    sig is the signature gained by evaluating the modemap condition
    elt is the slot number to get the implementation
    cond are runtime checks which are the results of evaluating the
    modemap condition

the following flags are used:
 $Coerce is NIL, if function selection is done which requires exact
   matches (e.g. for coercion functions)
 if $SubDom is true, then runtime checks have to be compiled
```

18.0.15 defun ofCategory

[identp p1040]
[ofCategory p417]
[hasCaty p418]
[$Subst p??]
[$hope p??]

— defun ofCategory —

```
(defun |ofCategory| (dom cat)
 (let (|$Subst| |$hope|)
 (declare (special |$Subst| |$hope|))
```

```
  (cond
   ((identp dom) nil)
   ((and (listp cat) (eq (car cat) '|Join|))
    (every #'(lambda (c) (|ofCategory| dom c)) (cdr cat)))
   (t (not (eq (|hasCaty| dom cat nil) '|failed|))))))
```

———-

18.0.16 defun isPartialMode

The isPartialMode function tests whether m contains `$EmptyMode`. The constant `$EmptyMode` evaluates to `|$EmptyMode|`. This constant is inserted in a modemap during compile time if the modemap is not yet complete. [contained p**??**]
[$EmptyMode p**??**]

— **defun isPartialMode** —

```
(defun |isPartialMode| (m)
 (declare (special |$EmptyMode|))
 (contained |$EmptyMode| m))
```

———-

18.0.17 defun hasCaty

This calls hasCat, which looks up a hashtable and returns:

```
 1. T, NIL or a (has x1 x2) condition, if cat is not parameterized
 2. a list of pairs (argument to cat,condition) otherwise
```

then the substitution sl is augmented, or the result is 'failed [hasAttSig p424]
[subCopy p**??**]
[constructSubst p432]
[hasSig p421]
[hasAtt p422]
[hasCat p**??**]
[opOf p**??**]
[kdr p**??**]
[mkDomPvar p430]
[domArg p420]
[augmentSub p**??**]
[domArg2 p420]
[unifyStruct p426]
[hasCaty1 p429]

[$domPvar p49]

— defun hasCaty —

```
(defun |hasCaty| (d cat sl)
 (let (x y S z cond sp dom zp s1 ncond i)
 (declare (special |$domPvar|))
  (cond
   ((and (consp cat) (eq (qcar cat) 'category) (consp (qcdr cat)))
     (|hasAttSig| d (|subCopy| (qcddr cat) (|constructSubst| d)) sl))
   ((and (consp cat) (eq (qcar cat) 'signature) (consp (qcdr cat))
         (consp (qcddr cat)) (eq (qcdddr cat) nil))
     (|hasSig| d (qcadr cat) (|subCopy| (qcaddr cat) (|constructSubst| d)) sl))
   ((and (consp cat) (eq (qcar cat) 'attribute)
         (consp (qcdr cat)) (eq (qcddr cat) nil))
     (|hasAtt| d (|subCopy| (qcadr cat) (|constructSubst| d)) sl))
   ((setq x (|hasCat| (|opOf| d) (|opOf| cat)))
    (cond
     ((setq y (kdr cat))
       (setq s (|constructSubst| d))
       (do ((next x (cdr next)) (endtest nil (null (eq s1 '|failed|))))
           ((or (atom next) endtest) nil)
         (setq z (caar next))
         (setq cond (cdar next))
         (setq sp
          (loop for item in s
           collect (cons (car item) (|mkDomPvar| (car item) (cdr item) z y))))
         (when |$domPvar|
           (setq i -1)
           (setq dom
            (cons (car d)
             (loop for arg in (rest d)
              collect (|domArg| arg (incf i) z y))))
           (setq sl (|augmentSub| |$domPvar| dom (copy sl))))
         (setq zp
          (loop for a in z
           collect (|domArg2| a s sp)))
         (setq s1 (|unifyStruct| y zp (copy sl)))
         (cond
          ((null (eq s1 '|failed|))
            (setq s1
             (cond
              ((atom cond) s1)
              (t
               (setq ncond (|subCopy| cond s))
               (cond
                ((and (consp ncond) (eq (qcar ncond) '|has|)
                      (consp (qcdr ncond)) (equal (qcadr ncond) d)
                      (consp (qcddr ncond)) (eq (qcdddr ncond) nil)
                      (equal (qcaddr ncond) cat))
```

```
                ’|failed|)
               (t (|hasCaty1| ncond s1)))))))
          (t nil)))
        s1)
     ((atom x) sl)
     (t
      (setq ncond (|subCopy| x (|constructSubst| d)))
      (cond
       ((and (consp ncond) (eq (qcar ncond) ’|has|) (consp (qcdr ncond))
             (equal (qcadr ncond) d) (consp (qcddr ncond))
             (eq (qcdddr ncond) nil) (equal (qcaddr ncond) cat))
        ’|failed|)
       (t (|hasCaty1| ncond sl))))))
   (t ’|failed|))))
```

———-

18.0.18 defun domArg

[$FormalMapVariableList p**??**]

— **defun domArg** —

```
(defun |domArg| (type i subs y)
 (let (p)
  (declare (special |$FormalMapVariableList|))
  (if (setq p (member (elt |$FormalMapVariableList| i) subs))
   (elt y (- (|#| subs) (|#| p)))
   type)))
```

———-

18.0.19 defun domArg2

[isSharpVar p876]
[subCopy p**??**]
[$domPvar p49]

— **defun domArg2** —

```
(defun |domArg2| (arg sl1 sl2)
 (declare (special |$domPvar|))
 (cond
  ((|isSharpVar| arg) (|subCopy| arg sl1))
  ((and (eq arg ’$) |$domPvar|) |$domPvar|)
```

```
  (t (|subCopy| arg sl2))))
```

———-

18.0.20 defun hasSig

The function hasSig tests whether domain dom has function foo with signature sig under substitution sl. [constructor? p**??**]
[cnstructSubst p**??**]
[assq p1044]
[getOperationAlistFromLisplib p**??**]
[hasCate p431]
[subCopy p**??**]
[hasSigAnd p423]
[hasSigOr p424]
[keyedSystemError p**??**]
[unifyStruct p426]
[$domPvar p49]

— **defun hasSig** —

```
(defun |hasSig| (dom foo sig sl)
 (let (|$domPvar| fun s0 p x cond s)
 (declare (special |$domPvar|))
  (cond
   ((setq fun (|constructor?| (car dom)))
     (setq s0 (|constructSubst| dom))
     (cond
      ((setq p (assq foo (|getOperationAlistFromLisplib| (car dom))))
        (do ((next (cdr p) (cdr next))
             (endtest nil (null (eq s '|failed|))))
            ((or (atom next) endtest) nil)
          (setq x (caar next))
          (setq cond (caddar next))
          (setq s
           (cond
            ((atom cond) (copy sl))
            ((and (consp cond) (eq (qcar cond) '|has|)
                  (consp (qcdr cond)) (consp (qcddr cond))
                  (eq (qcdr (qcddr cond)) nil))
              (|hasCate| (|subCopy| (qcadr cond) s0)
                         (|subCopy| (qcaddr cond) s0)
                         (copy sl)))
            ((and (consp cond)
                  (or (eq (qcar cond) 'and) (eq (qcar cond) '|and|)))
              (|hasSigAnd| (qcdr cond) s0 sl))
            ((and (consp cond)
```

```
                (or (eq (qcar cond) 'or) (eq (qcar cond) '|or|)))
            (|hasSigOr| (qcdr cond) s0 sl))
           (t
            (|keyedSystemError| 'S2GE0016
             (list "hasSig" "unexpected condition for signature")))))
         (unless (eq s '|failed|)
           (setq s (|unifyStruct| (|subCopy| x s0) sig s))))
       s)
     (t '|failed|)))
  (t '|failed|))))
```

———-

18.0.21 defun hasAtt

The hasAtt function tests whether dom has attribute att under sl needs s0 similar to hasSig.
[subCopy p**??**]
[getdatabase p1001]
[constructSubst p432]
[getInfovec p**??**]
[unifyStruct p426]
[hasCatExpression p425]
[$domPvar p49]

— **defun hasAtt** —

```
(defun |hasAtt| (dom att sl)
 (let (|$domPvar| fun atts u x cond s)
 (declare (special |$domPvar|))
  (cond
   ((setq fun (car dom))
    (cond
     ((setq atts
        (|subCopy| (getdatabase fun 'attributes) (|constructSubst| dom)))
       (cond
        ((consp (setq u (|getInfovec| (car dom))))
          (do ((next atts (cdr next))
               (endtest nil (null (eq s '|failed|))))
              ((or (atom next) endtest) nil)
            (setq x (caar next))
            (setq cond (cdar next))
            (setq s (|unifyStruct| x att (copy sl)))
            (cond
             ((and (null (atom cond)) (null (eq s '|failed|)))
               (setq s (|hasCatExpression| cond s)))))
          s)
        (t
```

```
        (do ((next atts (cdr next))
             (endtest nil (null (eq s '|failed|))))
            ((or (atom next) endtest) nil)
          (setq x (caar next))
          (setq cond (cadar next))
          (setq s (|unifyStruct| x att (copy sl)))
          (cond
           ((and (null (atom cond)) (null (eq s '|failed|)))
            (setq s (|hasCatExpression| cond s)))))
        s)))
    (t '|failed|)))
  (t '|failed|))))
```

———-

18.0.22 defun hasSigAnd

[hasCate p431]
[subCopy p??]
[keyedSystemError p??]

— **defun hasSigAnd** —

```
(defun |hasSigAnd| (andCls s0 sl)
 (let (sa dead)
  (setq sa '|failed|)
  (loop for cls in andCls
   do
    (when dead (return))
    (setq sa
     (cond
      ((atom cls) (copy sl))
      ((and (consp cls) (eq (qcar cls) '|has|) (consp (qcdr cls))
            (consp (qcddr cls)) (eq (qcdddr cls) nil))
        (|hasCate| (|subCopy| (qcadr cls) s0)
                   (|subCopy| (qcaddr cls) s0)
                   (copy sl)))
      (t
        (|keyedSystemError| 'S2GE0016
         (list  "hasSigAnd" "unexpected condition for signature")))))
    (when (eq sa '|failed|) (setq dead t)))
  sa))
```

———-

18.0.23 defun hasSigOr

[hasCate p431]
[hasSigAnd p423]
[keyedSystemError p**??**]

— **defun hasSigOr** —

```
(defun |hasSigOr| (orCls s0 sl)
 (let (sa found)
  (setq sa '|failed|)
  (loop for cls in orCls
   until found
   do
    (setq sa
     (cond
      ((atom cls) (copy sl))
      ((and (consp cls) (eq (qcar cls) '|has|) (consp (qcdr cls))
            (consp (qcddr cls)) (eq (qcdddr cls) nil))
        (|hasCate| (|subCopy| (qcadr cls) s0)
                   (|subCopy| (qcaddr cls) s0)
                   (copy sl)))
      ((and (consp cls)
            (or (eq (qcar cls) 'and) (eq (qcar cls) '|and|)))
        (|hasSigAnd| (qcdr cls) s0 sl))
      (t
       (|keyedSystemError| 'S2GE0016
        (list  "hasSigOr" "unexpected condition for signature")))))
    (unless (eq sa '|failed|) (setq found t)))
  sa))
```

————-

18.0.24 defun hasAttSig

The argument d is domain, x is a list of attributes and signatures. The result is an augmented SL, if d has x, 'failed otherwise. [hasAtt p422]
[hasSig p421]
[keyedSystemError p**??**]

— **defun hasAttSig** —

```
(defun |hasAttSig| (d x sl)
  (loop for y in x
   until (eq sl '|failed|)
   do
    (setq sl
```

```
    (cond
     ((and (consp y) (eq (qcar y) 'attribute)
           (consp (qcdr y)) (eq (qcddr y) nil))
       (|hasAtt| d (qcadr y) sl))
     ((and (consp y) (eq (qcar y) 'signature)
           (consp (qcdr y)) (consp (qcddr y)) (eq (qcdddr y) nil))
       (|hasSig| d (qcadr y) (qcaddr y) sl))
     (t
      (|keyedSystemError| 'S2GE0016
       (list  "hasAttSig" "unexpected form of unnamed category"))))))
  sl)
```

———-

18.0.25 defun hasCate1

[hasCate p431]
[$domPvar p49]

— **defun hasCate1** —

```
(defun |hasCate1| (dom cat sl domPvar)
 (let (|$domPvar|)
 (declare (special |$domPvar|))
  (setq |$domPvar| domPvar)
  (|hasCate| dom cat sl)))
```

———-

18.0.26 defun hasCatExpression

[hasCatExpression p425]
[hasCate p431]
[keyedSystemError p**??**]

— **defun hasCatExpression** —

```
(defun |hasCatExpression| (cond sl)
 (let (y)
  (cond
   ((and (consp cond) (eq (qcar cond) 'or))
    (when
     (let (result)
       (loop for x in (qcdr cond)
        do (setq result
```

```
           (or result
               (not (eq (setq y (|hasCatExpression| x sl)) '|failed|)))))
       result)
     y))
   ((and (consp cond) (eq (qcar cond) 'and))
    (when
     (let ((result t))
       (loop for x in (qcdr cond)
        do (setq result
             (and result
                  (not (eq (setq sl (|hasCatExpression| x sl)) '|failed|)))))
       result)
     sl))
   ((and (consp cond) (eq (qcar cond) '|has|)
         (consp (qcdr cond)) (consp (qcddr cond)) (eq (qcdddr cond) nil))
     (|hasCate| (qcadr cond) (qcaddr cond) sl))
   (t
    (|keyedSystemError| 'S2GE0016
     (list  "hasSig" "unexpected condition for attribute"))))))
```

———-

18.0.27 defun unifyStruct

[isPatternVar p**??**]
[unifyStructVar p427]
[unifyStruct p426]

— **defun unifyStruct** —

```
(defun |unifyStruct| (s1 s2 sl)
 (declare (special |$domPvar| |$hope| |$Coerce| |$Subst|))
  (cond
   ((equal s1 s2) sl)
   (t
    (when (and (consp s1) (eq (qcar s1) '|:|)
               (consp (qcdr s1)) (consp (qcddr s1)) (eq (qcdddr s1) nil))
       (setq s1 (qcadr s1)))
    (when (and (consp s2) (eq (qcar s2) '|:|)
               (consp (qcdr s2)) (consp (qcddr s2)) (eq (qcdddr s2) nil))
       (setq s2 (qcadr s2)))
    (when (and (null (atom s1)) (eq (car s1) '|#|))
       (setq s1 (length (cadr s1))))
    (when (and (null (atom s2)) (eq (car s2) '|#|))
       (setq s2 (length (cadr s2))))
    (cond
      ((equal s1 s2) sl)
```

```
((|isPatternVar| s1) (|unifyStructVar| s1 s2 sl))
((|isPatternVar| s2) (|unifyStructVar| s2 s1 sl))
((or (atom s1) (atom s2)) '|failed|)
(t
 (loop until (or (null s1) (null s2) (eq sl '|failed|))
  do
   (setq sl (|unifyStruct| (car s1) (car s2) sl))
   (setq s1 (cdr s1))
   (setq s2 (cdr s2)))
 (if (or s1 s2) '|failed| sl))))))
```

———-

18.0.28 defun unifyStructVar

The first argument is a pattern variable, which is not substituted by sl [contained p**??**]
[lassoc p**??**]
[unifyStruct p426]
[constructor? p**??**]
[subCopy p**??**]
[containsVars p428]
[canCoerce p**??**]
[resolveTT p**??**]
[isPatternVar p**??**]
[augmentSub p**??**]
[$domPvar p49]
[$Coerce p**??**]
[$Subst p**??**]
[$hope p**??**]

— **defun unifyStructVar** —

```
(defun |unifyStructVar| (v ss sl)
 (let (ps s1 s0 s ns0 ns1 s3)
  (declare (special |$domPvar| |$hope| |$Coerce| |$Subst|))
   (cond
    ((contained v ss) '|failed|)
    (t
     (setq ps (lassoc ss sl))
     (setq s1 (if ps ps ss))
     (cond
      ((or (setq s0 (lassoc v sl)) (setq s0 (lassoc v |$Subst|)))
        (setq s (|unifyStruct| s0 s1 (copy sl)))
        (cond
          ((eq s '|failed|)
           (cond
```

```
          ((and |$Coerce| (null (atom s0)) (|constructor?| (car s0)))
            (cond
             ((or (|containsVars| s0) (|containsVars| s1))
               (setq ns0 (|subCopy| s0 sl))
               (setq ns1 (|subCopy| s1 sl))
               (cond
                 ((or (|containsVars| ns0) (|containsVars| ns1))
                   (setq |$hope| t)
                   '|failed|)
                 (t
                  (cond
                   ((|canCoerce| ns0 ns1) (setq s3 s1))
                   ((|canCoerce| ns1 ns0) (setq s3 s0))
                   (t (setq s3 nil)))
                  (cond
                   (s3
                    (cond
                      ((not (equal s3 s0))
                        (setq sl (|augmentSub| v s3 sl))))
                    (cond
                     ((and (not (equal s3 s1)) (|isPatternVar| ss))
                        (setq sl (|augmentSub| ss s3 sl))))
                           sl)
                    (t '|failed|)))))
              (|$domPvar|
               (setq s3 (|resolveTT| s0 s1))
               (cond
                (s3
                 (cond
                  ((not (equal s3 s0))
                    (setq sl (|augmentSub| v s3 sl))))
                 (cond
                  ((and (not (equal s3 s1)) (|isPatternVar| ss))
                    (setq sl (|augmentSub| ss s3 sl))))
                 sl)
                (t '|failed|)))
              (t '|failed|)))
            (t '|failed|)))
          (t (|augmentSub| v ss s))))
      (t (|augmentSub| v ss sl)))))))
```

18.0.29 defun containsVars

The function containsVars tests whether term t contains a * variable. [isPatternVar p**??**] [containsVars1 p429]

— **defun containsVars** —

```
(defun |containsVars| (arg)
 (if (atom arg)
  (|isPatternVar| arg)
  (|containsVars1| arg)))
```

———-

18.0.30 defun containsVars1

The function containsVars1 tests whether term t contains a * variable. This is a recursive version, which works on a list. [isPatternVar p**??**]
[containsVars1 p429]

— **defun containsVars1** —

```
(defun |containsVars1| (arg)
 (let ((t1 (car arg)) (t2 (cdr arg)))
  (if (atom t1)
     (or (|isPatternVar| t1)
         (if (atom t2) (|isPatternVar| t2) (|containsVars1| t2)))
     (or (|containsVars1| t1)
         (if (atom t2)  (|isPatternVar| t2) (|containsVars1| t2))))))
```

———-

18.0.31 defun hasCaty1

The cond is either a (has a b) or an OR clause of such conditions. SL is augmented, if cond is true, otherwise the result is 'failed [hasCate p431]
[hasCaty1 p429]
[keyedSystemError p**??**]
[$domPvar p49]

— **defun hasCaty1** —

```
(defun |hasCaty1| (cond sl)
 (let (|$domPvar| a s)
 (declare (special |$domPvar|))
  (setq |$domPvar| nil)
   (cond
    ((and (consp cond) (eq (qcar cond) '|has|)
```

```
        (consp (qcdr cond)) (consp (qcddr cond)) (eq (qcdddr cond) nil))
   (|hasCate| (qcadr cond) (qcaddr cond) sl))
  ((and (consp cond) (EQ (qcar cond) 'and))
   (loop for x in (qcdr cond)
    while (not (eq s '|failed|))
    do
     (setq s
      (cond
       ((and (consp x) (eq (qcar x) '|has|)
             (consp (qcdr x)) (consp (qcddr x)) (eq (qcdr (qcddr x)) nil))
         (|hasCate| (qcadr x) (qcaddr x) sl))
       ((and (consp x) (eq (qcdr x) nil)
              (consp (qcar x)) (eq (qcaar x) '|has|)
              (consp (qcdar x)) (consp (qcddar x))
              (eq (qcdr (qcddar x)) nil))
         (|hasCate| a (qcaddar x) sl))
       (t (|hasCaty1| x sl)))))
   s)
  ((and (consp cond) (eq (qcar cond) 'or))
    (do ((next (qcdr cond) (cdr next)) (x nil)
         (nextitem nil (null (eq s '|failed|))))
        ((or (atom next)
             (progn (setq x (car next)) nil)
             nextitem)
             nil)
     (setq s
      (cond
       ((and (consp x) (eq (qcar x) '|has|)
             (consp (qcdr x)) (consp (qcddr x)) (eq (qcdddr x) nil))
         (|hasCate| (qcadr x) (qcaddr x) (copy sl)))
       ((and (consp x) (eq (qcdr x) nil) (consp (qcar x))
             (eq (qcaar x) '|has|) (consp (qcdar x)) (consp (qcddar x))
             (eq (qcdddar x) nil))
         (|hasCate| (qcadar x) (qcaddar x) (copy sl)))
       (t (|hasCaty1| x (copy sl))))))
   s)
  (t
   (|keyedSystemError| 'S2GE0016
    (list  "hasCaty1" "unexpected condition from category table"))))))
```

———-

18.0.32 defun mkDomPvar

[domArg p420]
[length p**??**]
[$FormalMapVariableList p**??**]

— **defun mkDomPvar** —

```
(defun |mkDomPvar| (p d subs y)
 (let (l)
 (declare (special |$FormalMapVariableList|))
  (if (setq l (member p |$FormalMapVariableList|))
   (|domArg| d (- (|#| |$FormalMapVariableList|) (|#| l)) subs y)
   d)))
```

———-

18.0.33 defun hasCate

[isPatternVar p**??**]
[hasCate1 p425]
[hasCateSpecial p432]
[containsVariables p**??**]
[subCopy p**??**]
[hasCaty p418]
[$EmptyMode p**??**]
[$Subst p**??**]
[$hope p**??**]

— **defun hasCate** —

```
(defun |hasCate| (dom cat sl)
 (let (nsl p s sl1)
 (declare (special |$hope| |$Subst| |$EmptyMode|))
  (cond
   ((equal dom |$EmptyMode|) nil)
   ((|isPatternVar| dom)
    (cond
     ((and (setq p (assq dom sl))
           (not (eq (setq nsl (|hasCate| (cdr p) cat sl)) '|failed|)))
      nsl)
     ((or (setq p (assq dom |$Subst|)) (setq p (assq dom sl)))
       (setq s (|hasCate1| (cdr p) cat sl dom))
       (cond
         ((null (eq s '|failed|)) s)
         (t (|hasCateSpecial| dom (cdr p) cat sl))))
     (t
      (when (not (eq sl '|failed|)) (setq |$hope| t))
      '|failed|)))
   (t
    (setq sl1
     (loop for item in sl
```

```
      when (null (|containsVariables| (cdr item)))
      collect item))
    (when sl1 (setq cat (|subCopy| cat sl1)))
    (|hasCaty| dom cat sl)))))
```

———-

18.0.34 defun constructSubst

[internl p**??**]
[stringimage p**??**]

— **defun constructSubst** —

```
(defun |constructSubst| (d)
 (let (sl (i 0))
  (setq sl (list (cons '$ d)))
  (when (listp d)
   (dolist (x (cdr d))
    (setq i (1+ i))
    (setq sl (cons (cons (internl "#" (stringimage i)) x) sl))))
  sl))
```

———-

18.0.35 defun hasCateSpecial

The variable v is a pattern variable, dom is its binding under `$Subst`. We try to change dom so that it has category cat under sl. The result is a substitution list or 'failed. [eqcar p**??**]
[isSubDomain p**??**]
[canCoerceFrom p**??**]
[containsVars p428]
[augmentSub p**??**]
[hasCate p431]
[hasCaty p418]
[hasCateSpecialNew p433]
[$Integer p**??**]
[$QuotientField p**??**]

— **defun hasCateSpecial** —

```
(defun |hasCateSpecial| (v dom cat sl)
 (let (arg d domp nsl)
 (declare (special |$Integer| |$QuotientField|))
```

```
  (cond
   ((and (consp dom) (eq (qcar dom) '|FactoredForm|)
         (consp (qcdr dom)) (eq (qcddr dom) nil))
     (setq arg (qcadr dom))
     (when (|isSubDomain| arg |$Integer|) (setq arg |$Integer|))
     (setq d (list '|FactoredRing| arg))
     (setq sl (|hasCate| arg '(|Ring|) (|augmentSub| v d sl)))
     (if (eq sl '|failed|)
      '|failed|
      (|hasCaty| d cat sl)))
   ((or (eqcar cat '|Field|) (eqcar cat '|DivisionRing|))
     (when (|isSubDomain| dom |$Integer|) (setq dom |$Integer|))
     (setq d (list |$QuotientField| dom))
     (|hasCaty| dom '(|IntegralDomain|) (|augmentSub| v d sl)))
   ((and (consp cat) (eq (qcar cat) '|PolynomialCategory|)
         (consp (qcdr cat)))
     (setq domp (cons '|Polynomial| (list (qcadr cat))))
     (and (or (|containsVars| (qcadr cat)) (|canCoerceFrom| dom domp))
              (|hasCaty| domp cat (|augmentSub| v domp sl))))
   ((|isSubDomain| dom |$Integer|)
     (setq nsl (|hasCate| |$Integer| cat (|augmentSub| v |$Integer| sl)))
     (if (eq nsl '|failed|)
      (|hasCateSpecialNew| v dom cat sl)
      (|hasCaty| |$Integer| cat nsl)))
   (t
     (|hasCateSpecialNew| v dom cat sl)))))
```

———-

18.0.36 defun hasCateSpecialNew

[member p1042]
[eqcar p**??**]
[augmentSub p**??**]
[defaultTargetFE p435]
[isEqualOrSubDomain p436]
[underDomainOf p**??**]
[hasCaty p418]
[$Integer p**??**]
[$ComplexInteger p**??**]
[$RationalNumber p**??**]

— **defun hasCateSpecialNew** —

```
(defun |hasCateSpecialNew| (v dom cat sl)
 (let (fe alg fefull d partialResult)
 (declare (special |$RationalNumber| |$ComplexInteger| |$Integer|))
```

```
(setq fe
 (|member| (qcar cat)
  '(|ElementaryFunctionCategory|
    |TrigonometricFunctionCategory|
    |ArcTrigonometricFunctionCategory|
    |HyperbolicFunctionCategory|
    |ArcHyperbolicFunctionCategory|
    |PrimitiveFunctionCategory|
    |SpecialFunctionCategory|
    |Evalable|
    |CombinatorialOpsCategory|
    |TranscendentalFunctionCategory|
    |AlgebraicallyClosedFunctionSpace|
    |ExpressionSpace|
    |LiouvillianFunctionCategory|
    |FunctionSpace|)))
(setq alg
 (|member| (qcar cat)
  '(|RadicalCategory|
    |AlgebraicallyClosedField|)))
(setq fefull
 (or fe alg (eqcar cat '|CombinatorialFunctionCategory|)))
(setq partialResult
 (cond
  ((or (eqcar dom '|Variable|) (eqcar dom '|Symbol|))
    (cond
     ((|member| (car cat)
       '(|SemiGroup|
         |AbelianSemiGroup|
         |Monoid|
         |AbelianGroup|
         |AbelianMonoid|
         |PartialDifferentialRing|
         |Ring|
         |InputForm|))
       (setq d (list '|Polynomial| |$Integer|))
       (|augmentSub| v d sl))
     ((eqcar cat '|Group|)
       (setq d (list '|Fraction| (list '|Polynomial| |$Integer|)))
       (|augmentSub| v d sl))
     (fefull
      (setq d (|defaultTargetFE| dom))
      (|augmentSub| v d sl))
     (t '|failed|)))
  ((|isEqualOrSubDomain| dom |$Integer|)
    (cond
     (fe
       (setq d (|defaultTargetFE| |$Integer|))
       (|augmentSub| v d sl))
     (alg
```

```
        (setq d '(|AlgebraicNumber|))
        (|augmentSub| v d sl))
       (t '|failed|)))
    ((equal (|underDomainOf| dom) |$ComplexInteger|)
      (setq d (|defaultTargetFE| |$ComplexInteger|))
      (|hasCaty| d cat (|augmentSub| v d sl)))
    ((and (equal dom |$RationalNumber|) alg)
      (setq d '(|AlgebraicNumber|))
      (|augmentSub| v d sl))
    (fefull
      (setq d (|defaultTargetFE| dom))
      (|augmentSub| v d sl))
    (t '|failed|)))
  (if (eq partialResult '|failed|)
    '|failed|
    (|hasCaty| d cat partialResult))))
```

———-

18.0.37 defun defaultTargetFE

[typeIsASmallInteger p**??**]
[isEqualOrSubDomain p436]
[ifcar p**??**]
[defaultTargetFE p435]
[$FunctionalExpression p**??**]
[$Integer p**??**]
[$Symbol p**??**]
[$RationalNumber p**??**]

— defun defaultTargetFE —

```
(defun |defaultTargetFE| (&rest dom)
 (let (a options)
 (declare (special |$FunctionalExpression| |$Integer| |$Symbol|
                   |$RationalNumber|))
  (setq a (car dom))
  (setq options (cdr dom))
  (cond
   ((or (and (consp a) (eq (qcar a) '|Variable|)
             (consp (qcdr a)) (eq (qcddr a) nil))
        (equal a |$RationalNumber|)
        (member (qcar a) (list (qcar |$Symbol|) '|RationalRadicals| '|Pi|))
        (|typeIsASmallInteger| a)
        (|isEqualOrSubDomain| a |$Integer|)
        (equal a '(|AlgebraicNumber|)))
     (if (ifcar options)
```

```
      (list |$FunctionalExpression| (list '|Complex| |$Integer|))
      (list |$FunctionalExpression| |$Integer|)))
   ((and (consp a) (eq (qcar a) '|Complex|)
         (consp (qcdr a)) (eq (qcddr a) nil))
     (|defaultTargetFE| (qcadr a) t))
   ((and (consp a) (consp (qcdr a)) (eq (qcddr a) nil)
         (member (qcar a) '(|Polynomial| |RationalFunction| |Fraction|)))
     (|defaultTargetFE| (qcadr a) (ifcar options)))
   ((and (consp a) (equal (qcar a) |$FunctionalExpression|)
         (consp (qcdr a)) (eq (qcddr a) nil))
     a)
   ((ifcar options)
     (list |$FunctionalExpression| (list '|Complex| a)))
   (t
     (list |$FunctionalExpression| a)))))
```

———-

18.0.38 defun isEqualOrSubDomain

[isSubDomain p**??**]

— **defun isEqualOrSubDomain** —

```
(defun |isEqualOrSubDomain| (d1 d2)
  (or (equal d1 d2)
      (|isSubDomain| d1 d2)
      (and (atom d1)
           (or (and (consp d2) (eq (qcar d2) '|Variable|)
                    (consp (qcdr d2)) (eq (qcddr d2) nil)
                    (equal (qcadr d2) d1))
               (and (consp d2) (eq (qcdr d2) nil)
                    (equal (qcar d2) d1))))
      (and (atom d2)
           (or (and (consp d1) (eq (qcar d1) '|Variable|)
                    (consp (qcdr d1)) (eq (qcddr d1) nil)
                    (equal (qcadr d1) d2))
               (and (consp d1) (eq (qcdr d1) nil)
                    (equal (qcar d1) d2))))))
```

———-

Chapter 19

System Command Handling

The system commands are the top-level commands available in Axiom that can all be invoked by prefixing the symbol with a closed-paren. Thus, to see they copyright you type:

```
)copyright
```

New commands need to be added to this table. The command invoked will be the first entry of the pair and the "user level" of the command will be the second entry.

See:

- The "abbreviations" (20.2.1 p 479) command
- The "boot" (5.1.8 p 25) command
- The "browse" (**??** p **??**) command
- The "cd" (**??** p **??**) command
- The "clear" (24.3.1 p 495) command
- The "close" (25.2.2 p 506) command
- The "compile" (**??** p **??**) command
- The "copyright" (27.2.1 p 518) command
- The "credits" (28.3.1 p 521) command
- The "display" (30.2.1 p 531) command
- The "edit" (31.2.1 p 540) command
- The "fin" (32.1.1 p 544) command

- The “frame” (33.5.16 p 561) command
- The “help” (34.2.1 p 568) command
- The “history” (35.4.7 p 578) command
- The “lisp” (**??** p **??**) command
- The “library” (65.1.34 p 1005) command
- The “load” (39.1.1 p 625) command
- The “ltrace” (40.1.1 p 628) command
- The “pquit” (41.2.1 p 630) command
- The “quit” (42.2.1 p 634) command
- The “read” (43.1.1 p 638) command
- The “savesystem” (44.1.1 p 642) command
- The “set” (45.36.1 p 799) command
- The “show” (46.1.1 p 806) command
- The “spool” (**??** p **??**) command
- The “summary” (48.1.1 p 822) command
- The “synonym” (49.1.1 p 824) command
- The “system” (**??** p **??**) command
- The “trace” (51.1.7 p 837) command
- The “trademark” (27.2.2 p 519) command
- The “undo” (52.4.6 p 912) command
- The “what” (53.1.2 p 929) command
- The “with” (54.1.1 p 937) command
- The “workfiles” (55.1.1 p 939) command
- The “zsystemdevelopment” (56.1.1 p 943) command

19.1 Variables Used

19.1.1 defvar $systemCommands

— initvars —

```
(defvar |$systemCommands| nil)
```

————-

— postvars —

```
(eval-when (eval load)
 (setq |$systemCommands|
 '(
   (|abbreviations|                . |compiler|   )
   (|boot|                         . |development|)
   (|browse|                       . |development|)
   (|cd|                           . |interpreter|)
   (|clear|                        . |interpreter|)
   (|close|                        . |interpreter|)
   (|compiler|                     . |compiler|   )
   (|copyright|                    . |interpreter|)
   (|credits|                      . |interpreter|)
   (|describe|                     . |interpreter|)
   (|display|                      . |interpreter|)
   (|edit|                         . |interpreter|)
   (|fin|                          . |development|)
   (|frame|                        . |interpreter|)
   (|help|                         . |interpreter|)
   (|history|                      . |interpreter|)
   (|lisp|                         . |development|)
   (|library|                      . |interpreter|)
   (|load|                         . |interpreter|)
   (|ltrace|                       . |interpreter|)
   (|pquit|                        . |interpreter|)
   (|quit|                         . |interpreter|)
   (|read|                         . |interpreter|)
   (|savesystem|                   . |interpreter|)
   (|set|                          . |interpreter|)
   (|show|                         . |interpreter|)
   (|spool|                        . |interpreter|)
   (|summary|                      . |interpreter|)
   (|synonym|                      . |interpreter|)
   (|system|                       . |interpreter|)
   (|trace|                        . |interpreter|)
```

```
   (|trademark|                    . |interpreter|)
   (|undo|                         . |interpreter|)
   (|what|                         . |interpreter|)
   (|with|                         . |interpreter|)
   (|workfiles|                    . |development|)
   (|zsystemdevelopment|           . |interpreter|)
 )))
```

———-

19.1.2 defvar $syscommands

This table is used to look up a symbol to see if it might be a command.

— **initvars** —

```
(defvar $syscommands nil)
```

———-

— **postvars** —

```
(eval-when (eval load)
 (setq $syscommands (mapcar #'car |$systemCommands|)))
```

———-

19.1.3 defvar $noParseCommands

This is a list of the commands which have their arguments passed verbatim. Certain functions, such as the lisp function need to be able to handle all kinds of input that will not be acceptable to the interpreter.

— **initvars** —

```
(defvar |$noParseCommands| nil)
```

———-

— **postvars** —

```
(eval-when (eval load)
 (setq |$noParseCommands|
```

```
'(|boot| |copyright| |credits| |fin| |lisp| |pquit| |quit|
  |synonym| |system| |trademark| )))
```

19.2 Functions

19.2.1 defun handleNoParseCommands

The system commands given by the global variable `$noParseCommands` require essentially no preprocessing/parsing of their arguments. Here we dispatch the functions which implement these commands.

There are four standard commands which receive arguments

- boot
- lisp
- synonym
- system

There are six standard commands which do not receive arguments –

- quit
- fin
- pquit
- credits
- copyright
- trademark

As these commands do not necessarily exhaust those mentioned in `$noParseCommands`, we provide a generic dispatch based on two conventions: commands which do not require an argument name themselves, those which do have their names prefixed by "np". This makes it possible to dynamically define new system commands provided you handle the argument parsing.

19.2.2 defun Handle a top level command

[concat p1041]
[expand-tabs p??]
[processSynonyms p33]
[substring p??]
[getFirstWord p465]
[unAbbreviateKeyword p465]
[member p1042]
[handleNoParseCommands p441]
[splitIntoOptionBlocks p443]
[handleTokensizeSystemCommands p443]
[handleParsedSystemCommands p464]
[$tokenCommands p471]
[$noParseCommands p440]
[line p??]

— defun doSystemCommand —

```
(defun |doSystemCommand| (string)
 (let (line tok unab optionList)
 (declare (special line |$tokenCommands| |$noParseCommands|))
  (setq string (concat ")" (expand-tabs string)))
  (setq line string)
  (|processSynonyms|)
  (setq string line)
  (setq string (substring string 1 nil))
  (cond
   ((string= string "") nil)
   (t
    (setq tok (|getFirstWord| string))
    (cond
     (tok
      (setq unab (|unAbbreviateKeyword| tok))
      (cond
       ((|member| unab |$noParseCommands|)
        (|handleNoParseCommands| unab string))
       (t
        (setq optionList (|splitIntoOptionBlocks| string))
        (cond
         ((|member| unab |$tokenCommands|)
          (|handleTokensizeSystemCommands| unab optionList))
         (t
          (|handleParsedSystemCommands| unab optionList)
          nil)))))
     (t nil))))))
```

———-

19.2.3 defun Split block into option block

[stripSpaces p467]

— **defun splitIntoOptionBlocks** —

```
(defun |splitIntoOptionBlocks| (str)
 (let (inString block (blockStart 0) (parenCount 0) blockList)
  (dotimes (i (1- (|#| str)))
   (cond
    ((char= (elt str i) #\" ) (setq inString (null inString)))
    (t
     (when (and (char= (elt str i) #\( ) (null inString))
       (incf parenCount))
     (when (and (char= (elt str i) #\) ) (null inString))
       (decf parenCount))
     (when
      (and (char= (elt str i) #\) )
           (null inString)
           (= parenCount -1))
        (setq block (|stripSpaces| (subseq str blockStart i)))
        (setq blockList (cons block blockList))
        (setq blockStart (1+ i))
        (setq parenCount 0)))))
  (setq blockList (cons (|stripSpaces| (subseq str blockStart)) blockList))
  (nreverse blockList)))
```

————-

19.2.4 defun Tokenize a system command

[dumbTokenize p463]
[tokTran p463]
[systemCommand p444]

— **defun handleTokensizeSystemCommands** —

```
(defun |handleTokensizeSystemCommands| (unabr optionList)
 (declare (ignore unabr))
 (let (parcmd)
  (setq optionList (mapcar #'(lambda (x) (|dumbTokenize| x)) optionList))
  (setq parcmd
    (mapcar #'(lambda (opt) (mapcar #'(lambda (tok) (|tokTran| tok)) opt))
     optionLIst))
  (when parcmd (|systemCommand| parcmd))))
```

————-

19.2.5 defun Handle system commands

You can type ")?" and see trivial help information. You can type ")? compile" and see compiler related information [selectOptionLC p475]
[helpSpad2Cmd p568]
[selectOption p475]
[commandsForUserLevel p444]
[$options p**??**]
[$e p**??**]
[$systemCommands p439]
[$syscommands p440]
[$CategoryFrame p**??**]

— **defun systemCommand** —

```
(defun |systemCommand| (cmd)
 (let (|$options| |$e| op argl options fun)
 (declare (special |$options| |$e| |$systemCommands| $syscommands
                   |$CategoryFrame|))
  (setq op (caar cmd))
  (setq argl (cdar cmd))
  (setq options (cdr cmd))
  (setq |$options| options)
  (setq |$e| |$CategoryFrame|)
  (setq fun (|selectOptionLC| op $syscommands '|commandError|))
  (if (and argl (eq (elt argl 0) '?) (not (eq fun '|synonym|)))
   (|helpSpad2Cmd| (cons fun nil))
   (progn
    (setq fun
     (|selectOption| fun (|commandsForUserLevel| |$systemCommands|)
                         '|commandUserLevelError|))
    (funcall fun argl)))))
```

———-

19.2.6 defun Select commands matching this user level

The `$UserLevel` variable contains one of three values: `compiler`, `development`, or `interpreter`. This variable is used to select a subset of commands from the list stored in `$systemCommands`, representing all of the commands that are valid for this level. [satisfiesUserLevel p447]

— **defun commandsForUserLevel** —

```
(defun |commandsForUserLevel| (arg)
 (let (c)
  (dolist (pair arg)
```

```
   (when (|satisfiesUserLevel| (cdr pair))
    (setq c (cons (car pair) c))))
  (nreverse c)))
```

———-

19.2.7 defun No command begins with this string

[commandErrorMessage p445]

— **defun commandError** —

```
(defun |commandError| (x u)
 (|commandErrorMessage| '|command| x u))
```

———-

19.2.8 defun No option begins with this string

[commandErrorMessage p445]

— **defun optionError** —

```
(defun |optionError| (x u)
 (|commandErrorMessage| '|option| x u))
```

———-

19.2.9 defvar $oldline

— **initvars** —

```
(defvar $oldline nil "used to output command lines")
```

———-

19.2.10 defun No command/option begins with this string

[commandAmbiguityError p448]
[sayKeyedMsg p327]

[terminateSystemCommand p448]
[$oldline p445]
[line p??]

— **defun commandErrorMessage** —

```
(defun |commandErrorMessage| (kind x u)
 (declare (special $oldline line))
  (setq $oldline line)
  (if u
   (|commandAmbiguityError| kind x u)
   (progn
    (|sayKeyedMsg| 'S2IZ0008 (list kind x))
    (|terminateSystemCommand|))))
```

———-

19.2.11 defun Option not available at this user level

[userLevelErrorMessage p446]

— **defun optionUserLevelError** —

```
(defun |optionUserLevelError| (x u)
 (|userLevelErrorMessage| '|option| x u))
```

———-

19.2.12 defun Command not available at this user level

[userLevelErrorMessage p446]

— **defun commandUserLevelError** —

```
(defun |commandUserLevelError| (x u)
 (|userLevelErrorMessage| '|command| x u))
```

———-

19.2.13 defun Command not available error message

[commandAmbiguityError p448]
[sayKeyedMsg p327]

[terminateSystemCommand p448]
[$UserLevel p798]

— **defun userLevelErrorMessage** —

```
(defun |userLevelErrorMessage| (kind x u)
 (declare (special |$UserLevel|))
 (if u
  (|commandAmbiguityError| kind x u)
  (progn
   (|sayKeyedMsg| 'S2IZ0007 (list |$UserLevel| kind))
   (|terminateSystemCommand|))))
```

———-

19.2.14 defun satisfiesUserLevel

[$UserLevel p798]

— **defun satisfiesUserLevel 0** —

```
(defun |satisfiesUserLevel| (x)
 (declare (special |$UserLevel|))
 (cond
  ((eq x '|interpreter|) t)
  ((eq |$UserLevel| '|interpreter|) nil)
  ((eq x '|compiler|) t)
  ((eq |$UserLevel| '|compiler|) nil)
  (t t)))
```

———-

19.2.15 defun hasOption

[stringPrefix? p**??**]
[pname p1039]

— **defun hasOption** —

```
(defun |hasOption| (al opt)
 (let ((optPname (pname opt)) found)
  (loop for pair in al do
    (when (|stringPrefix?| (pname (car pair)) optPname) (setq found pair))
    until found)
  found))
```

———-

19.2.16 defun terminateSystemCommand

[tersyscommand p448]

— **defun terminateSystemCommand** —

```
(defun |terminateSystemCommand| nil (tersyscommand))
```

———-

19.2.17 defun Terminate a system command

[spadThrow p??]

— **defun tersyscommand** —

```
(defun tersyscommand ()
  (fresh-line)
  (setq chr 'endoflinechr)
  (setq tok 'end_unit)
  (|spadThrow|))
```

———-

19.2.18 defun commandAmbiguityError

[sayKeyedMsg p327]
[sayMSG p329]
[bright p??]
[terminateSystemCommand p448]

— **defun commandAmbiguityError** —

```
(defun |commandAmbiguityError| (kind x u)
 (|sayKeyedMsg| 's2iz0009 (list kind x))
 (dolist (a u) (|sayMSG| (cons "     " (|bright| a))))
 (|terminateSystemCommand|))
```

———-

19.2.19 defun getParserMacroNames

The `$pfMacros` is a list of all of the user-defined macros. [$pfMacros p98]

— **defun getParserMacroNames 0** —

```
(defun |getParserMacroNames| ()
 (declare (special |$pfMacros|))
 (remove-duplicates (mapcar #'car |$pfMacros|)))
```

———-

19.2.20 defun clearParserMacro

Note that if a macro is defined twice this will clear the last instance. Thus:

```
  a ==> 3
  a ==> 4
  )d macros
  a ==> 4
  )clear prop a
  )d macros
  a ==> 3
  )clear prop a
  )d macros
  nil
```

[ifcdr p**??**]
[assoc p**??**]
[remalist p**??**]
[$pfMacros p98]

— **defun clearParserMacro** —

```
(defun |clearParserMacro| (macro)
 (declare (special |$pfMacros|))
 (when (ifcdr (|assoc| macro |$pfMacros|))
  (setq |$pfMacros| (remalist |$pfMacros| macro))))
```

———-

19.2.21 defun displayMacro

[isInterpMacro p**??**]
[sayBrightly p**??**]

[bright p??]
[strconc p??]
[object2String p??]
[mathprint p??]
[$op p??]

— **defun displayMacro** —

```
(defun |displayMacro| (name)
 (let (|$op| m body args)
 (declare (special |$op|))
  (setq m (|isInterpMacro| name))
  (cond
   ((null m)
    (|sayBrightly|
     (cons "  " (append (|bright| name)
                        (cons "is not an interpreter macro." nil)))))
   (t
    (setq |$op| (strconc "macro " (|object2String| name)))
    (setq args (car m))
    (setq body (cdr m))
    (setq args
     (cond
      ((null args) nil)
      ((null (cdr args)) (car args))
      (t (cons '|Tuple| args))))
    (|mathprint| (cons 'map (cons (cons args body) nil)))))))
```

———-

19.2.22 defun displayWorkspaceNames

[getInterpMacroNames p??]
[getParserMacroNames p449]
[sayMessage p??]
[msort p??]
[getWorkspaceNames p451]
[sayAsManyPerLineAsPossible p??]
[sayBrightly p??]
[setdifference p??]

— **defun displayWorkspaceNames** —

```
(defun |displayWorkspaceNames| ()
 (let (pmacs names imacs)
  (setq imacs (|getInterpMacroNames|))
```

```
  (setq pmacs (|getParserMacroNames|))
  (|sayMessage| "Names of User-Defined Objects in the Workspace:")
  (setq names (msort (append (|getWorkspaceNames|) pmacs)))
  (if names
   (|sayAsManyPerLineAsPossible| (mapcar #'|object2String| names))
   (|sayBrightly| "   * None *"))
  (setq imacs (setdifference imacs pmacs))
  (when imacs
   (|sayMessage| "Names of System-Defined Objects in the Workspace:")
   (|sayAsManyPerLineAsPossible| (mapcar #'|object2String| imacs)))))
```

———-

19.2.23 defun getWorkspaceNames

```
;getWorkspaceNames() ==
;  NMSORT [n for [n,:.] in CAAR $InteractiveFrame |
;    (n ^= "--macros--" and n^= "--flags--")]
```

[seq p??]
[nmsort p??]
[exit p??]
[$InteractiveFrame p??]

— **defun getWorkspaceNames** —

```
(defun |getWorkspaceNames| ()
 (PROG (n)
  (declare (special |$InteractiveFrame|))
    (return
      (seq (nmsort (PROG (G166322)
                     (setq G166322 NIL)
                     (RETURN
                       (DO ((G166329 (CAAR |$InteractiveFrame|)
                                (CDR G166329))
                            (G166313 NIL))
                           ((OR (ATOM G166329)
                                (PROGN
                                  (SETQ G166313 (CAR G166329))
                                  NIL)
                                (PROGN
                                  (PROGN
                                    (setq n (CAR G166313))
                                    G166313)
                                  NIL))
                            (NREVERSE0 G166322))
                         (SEQ (EXIT (COND
```

```
((AND (not (eq n '|--macros--|))
  (not (eq n '|--flags--|)))
 (SETQ G166322
  (CONS n G166322)))))))))))))))
```

———-

19.2.24 defun fixObjectForPrinting

The `$msgdbPrims` variable is set to:

```
(|%b| |%d| |%l| |%i| |%u| %U |%n| |%x| |%ce| |%rj|
 "%U" "%b" "%d" "%l" "%i" "%u" "%U" "%n" "%x" "%ce" "%rj")
```

[object2Identifier p**??**]
[member p1042]
[strconc p**??**]
[pname p1039]
[$msgdbPrims p325]

— defun fixObjectForPrinting —

```
(defun |fixObjectForPrinting| (v)
 (let (vp)
 (declare (special |$msgdbPrims|))
  (setq vp (|object2Identifier| v))
  (cond
   ((eq vp '%) "\\%")
   ((|member| vp |$msgdbPrims|) (strconc "\\" (pname vp)))
   (t v))))
```

———-

19.2.25 defun displayProperties,sayFunctionDeps

```
;displayProperties(option,l) ==
;  $dependentAlist : local := nil
;  $dependeeAlist  : local := nil
;  [opt,:vl]:= (l or ['properties])
;  imacs := getInterpMacroNames()
;  pmacs := getParserMacroNames()
;  macros := REMDUP append(imacs, pmacs)
;  if vl is ['all] or null vl then
;    vl := MSORT append(getWorkspaceNames(),macros)
;  if $frameMessages then sayKeyedMsg("S2IZ0065",[$interpreterFrameName])
```

```
;  null vl =>
;    null $frameMessages => sayKeyedMsg("S2IZ0066",NIL)
;    sayKeyedMsg("S2IZ0067",[$interpreterFrameName])
;  interpFunctionDepAlists()
;  for v in vl repeat
;    isInternalMapName(v) => 'iterate
;    pl := getIProplist(v)
;    option = 'flags =>     getAndSay(v,"flags")
;    option = 'value =>     displayValue(v,getI(v,'value),nil)
;    option = 'condition => displayCondition(v,getI(v,"condition"),nil)
;    option = 'mode =>      displayMode(v,getI(v,'mode),nil)
;    option = 'type =>      displayType(v,getI(v,'value),nil)
;    option = 'properties =>
;      v = "--flags--" => nil
;      pl is [ ['cacheInfo,:.],:.] => nil
;      v1 := fixObjectForPrinting(v)
;      sayMSG ['"Properties of",:bright prefix2String v1,'":"]
;      null pl =>
;        v in pmacs =>
;            sayMSG '"   This is a user-defined macro."
;            displayParserMacro v
;        isInterpMacro v =>
;            sayMSG '"   This is a system-defined macro."
;            displayMacro v
;        sayMSG '"   none"
;      propsSeen:= nil
;      for [prop,:val] in pl | ^MEMQ(prop,propsSeen) and val repeat
;        prop in '(alias generatedCode IS_-GENSYM mapBody localVars) =>
;          nil
;        prop = 'condition =>
;          displayCondition(prop,val,true)
;        prop = 'recursive =>
;          sayMSG '"   This is recursive."
;        prop = 'isInterpreterFunction =>
;          sayMSG '"   This is an interpreter function."
;          sayFunctionDeps v where
;            sayFunctionDeps x ==
;              if dependents := GETALIST($dependentAlist,x) then
;                null rest dependents =>
;                  sayMSG ['"   The following function or rule ",
;                    '"depends on this:",:bright first dependents]
;                sayMSG
;                  '"   The following functions or rules depend on this:"
;                msg := ["%b",'"     "]
;                for y in dependents repeat msg := ['" ",y,:msg]
;                sayMSG [:nreverse msg,"%d"]
;              if dependees := GETALIST($dependeeAlist,x) then
;                null rest dependees =>
;                  sayMSG ['"   This depends on the following function ",
;                    '"or rule:",:bright first dependees]
```

```
;                sayMSG
;                  '"   This depends on the following functions or rules:"
;                msg := ["%b",'"     "]
;                for y in dependees repeat msg := ['" ",y,:msg]
;                sayMSG [:nreverse msg,"%d"]
;        prop = 'isInterpreterRule =>
;          sayMSG '"   This is an interpreter rule."
;          sayFunctionDeps v
;        prop = 'localModemap =>
;          displayModemap(v,val,true)
;        prop = 'mode =>
;          displayMode(prop,val,true)
;        prop = 'value =>
;          val => displayValue(v,val,true)
;        sayMSG ['"   ",prop,'":  ",val]
;        propsSeen:= [prop,:propsSeen]
;    sayKeyedMsg("S2IZ0068",[option])
;  terminateSystemCommand()
```

[seq p**??**]
[getalist p**??**]
[exit p**??**]
[sayMSG p329]
[bright p**??**]
[$dependeeAlist p**??**]
[$dependentAlist p**??**]

— defun displayProperties,sayFunctionDeps —

```
(defun |displayProperties,sayFunctionDeps| (x)
 (prog (dependents dependees msg)
 (declare (special |$dependeeAlist| |$dependentAlist|))
 (return
  (seq
   (if (setq dependents (getalist |$dependentAlist| x))
    (seq
     (if (null (cdr dependents))
      (exit
       (|sayMSG| (cons "   The following function or rule "
                   (cons "depends on this:" (|bright| (car dependents)))))))
     (|sayMSG| "   The following functions or rules depend on this:")
     (setq msg (cons '|%b| (cons "     " nil)))
     (do ((G166397 dependents (cdr G166397)) (y nil))
         ((or (atom G166397) (progn (setq y (car G166397)) nil)) nil)
       (seq (exit (setq msg (cons " " (cons y msg))))))
     (exit (|sayMSG| (append (nreverse msg) (cons '|%d| nil)))))
    nil)
   (exit
    (if (setq dependees (getalist |$dependeeAlist| x))
```

```
     (seq
      (if (null (cdr dependees))
       (exit
        (|sayMSG| (cons "   This depends on the following function "
                    (cons "or rule:" (|bright| (car dependees)))))))
      (|sayMSG| "   This depends on the following functions or rules:")
      (setq msg (cons '|%b| (cons "     " nil)))
      (do ((G166406 dependees (cdr G166406)) (y nil))
          ((or (atom G166406) (progn (setq y (car G166406)) nil)) nil)
        (seq (exit (setq msg (cons " " (cons y msg))))))
      (exit (|sayMSG| (append (nreverse msg) (cons '|%d| nil)))))
     nil))))))
```

———-

19.2.26 defun displayValue

[sayMSG p329]
[fixObjectForPrinting p452]
[pname p1039]
[objValUnwrap p??]
[objMode p??]
[displayRule p??]
[strconc p??]
[prefix2String p??]
[objMode p??]
[getdatabase p1001]
[concat p1041]
[form2String p??]
[mathprint p??]
[outputFormat p??]
[objMode p??]
[$op p??]
[$EmptyMode p??]

— defun displayValue —

```
(defun |displayValue| (|$op| u omitVariableNameIfTrue)
 (declare (special |$op|))
 (let (expr op rhs label labmode)
 (declare (special |$EmptyMode|))
  (if (null u)
   (|sayMSG|
    (list '|   Value of | (|fixObjectForPrinting| (pname |$op|)) ":  (none)"))
   (progn
    (setq expr (|objValUnwrap| u))
```

```
    (if (or (and (consp expr) (progn (setq op (qcar expr)) t) (eq op 'map))
            (equal (|objMode| u) |$EmptyMode|))
     (|displayRule| |$op| expr)
     (progn
       (cond
        (omitVariableNameIfTrue
          (setq rhs "):  ")
          (setq label "Value (has type "))
        (t
         (setq rhs ":  ")
         (setq label (strconc "Value of " (pname |$op|) ": "))))
      (setq labmode (|prefix2String| (|objMode| u)))
      (when (atom labmode) (setq labmode (list labmode)))
      (if (eq (getdatabase expr 'constructorkind) '|domain|)
       (|sayMSG| (|concat| "   " label labmode rhs (|form2String| expr)))
       (|mathprint|
        (cons 'concat
         (cons label
          (append labmode
           (cons rhs
            (cons (|outputFormat| expr (|objMode| u)) nil)))))))
      nil))))
```

———-

19.2.27 defun displayType

[sayMSG p329]
[fixObjectForPrinting p452]
[pname p1039]
[prefix2String p??]
[objMode p??]
[concat p1041]
[$op p??]

— defun displayType —

```
(defun |displayType| (|$op| u omitVariableNameIfTrue)
 (declare (special |$op|) (ignore omitVariableNameIfTrue))
 (let (type)
  (if (null u)
   (|sayMSG|
    (list  "   Type of value of " (|fixObjectForPrinting| (pname |$op|))
           ":  (none)"))
   (progn
    (setq type (|prefix2String| (|objMode| u)))
    (when (atom type) (setq type (list type)))
```

```
    (|sayMSG|
     (|concat|
      (cons "   Type of value of "
       (cons (|fixObjectForPrinting| (pname |$op|))
        (cons ": " type)))))
     nil))))
```

———-

19.2.28 defun getAndSay

[getI p??]
[sayMSG p329]

— **defun getAndSay** —

```
(defun |getAndSay| (v prop)
 (let (val)
  (if (setq val (|getI| v prop))
   (|sayMSG| (cons '|    | (cons val (cons '|%l| nil))))
   (|sayMSG| (cons '|    none| (cons '|%l| nil))))))
```

———-

19.2.29 defun displayProperties

[getInterpMacroNames p??]
[getParserMacroNames p449]
[remdup p??]
[qcdr p??]
[qcar p??]
[msort p??]
[getWorkspaceNames p451]
[sayKeyedMsg p327]
[interpFunctionDepAlists p461]
[isInternalMapName p??]
[getIProplist p??]
[getAndSay p457]
[displayValue p455]
[getI p??]
[displayCondition p461]
[displayMode p462]
[displayType p456]
[fixObjectForPrinting p452]

[sayMSG p329]
[bright p??]
[prefix2String p??]
[member p1042]
[displayParserMacro p460]
[isInterpMacro p??]
[displayMacro p449]
[displayProperties,sayFunctionDeps p452]
[displayModemap p462]
[exit p??]
[seq p??]
[terminateSystemCommand p448]
[$dependentAlist p??]
[$dependeeAlist p??]
[$frameMessages p732]
[$interpreterFrameName p??]

— defun displayProperties —

```
(defun |displayProperties| (option al)
 (let (|$dependentAlist| |$dependeeAlist| tmp1 opt imacs pmacs macros vl pl
       tmp2 vone prop val propsSeen)
 (declare (special |$dependentAlist| |$dependeeAlist| |$frameMessages|
                   |$interpreterFrameName|))
  (setq |$dependentAlist| nil)
  (setq |$dependeeAlist| nil)
  (setq tmp1 (or al (cons '|properties| nil)))
  (setq opt (car tmp1))
  (setq vl (cdr tmp1))
  (setq imacs (|getInterpMacroNames|))
  (setq pmacs (|getParserMacroNames|))
  (setq macros (remdup (append imacs pmacs)))
  (when (or
         (and (consp vl) (eq (qcdr vl) nil) (eq (qcar vl) '|all|))
         (null vl))
    (setq vl (msort (append (|getWorkspaceNames|) macros))))
   (when |$frameMessages|
    (|sayKeyedMsg| 'S2IZ0065 (cons |$interpreterFrameName| nil)))
   (cond
    ((null vl)
     (if (null |$frameMessages|)
      (|sayKeyedMsg| 'S2IZ0066 nil))
      (|sayKeyedMsg| 'S2IZ0067 (cons |$interpreterFrameName| nil)))
    (t
     (|interpFunctionDepAlists|)
     (do ((G166440 vl (cdr G166440)) (v nil))
         ((or (atom G166440) (progn (setq v (car G166440)) nil)) nil)
       (seq (exit
```

```
(cond
 ((|isInternalMapName| v) '|iterate|)
 (t
  (setq pl (|getIProplist| v))
  (cond
   ((eq option '|flags|)
    (|getAndSay| v '|flags|))
   ((eq option '|value|)
    (|displayValue| v (|getI| v '|value|) nil))
   ((eq option '|condition|)
    (|displayCondition| v (|getI| v '|condition|) nil))
   ((eq option '|mode|)
    (|displayMode| v (|getI| v '|mode|) nil))
   ((eq option '|type|)
    (|displayType| v (|getI| v '|value|) nil))
   ((eq option '|properties|)
    (cond
     ((eq v '|--flags--|)
       nil)
     ((and (consp pl)
           (progn
            (setq tmp2 (qcar pl))
            (and (consp tmp2) (eq (qcar tmp2) '|cacheInfo|))))
      nil)
     (t
      (setq vone (|fixObjectForPrinting| v))
      (|sayMSG|
       (cons "Properties of"
        (append (|bright| (|prefix2String| vone)) (cons ":" nil))))
      (cond
       ((null pl)
        (cond
         ((|member| v pmacs)
          (|sayMSG| "   This is a user-defined macro.")
          (|displayParserMacro| v))
         ((|isInterpMacro| v)
          (|sayMSG| "   This is a system-defined macro.")
          (|displayMacro| v))
         (t
          (|sayMSG| "   none"))))
       (t
        (setq propsSeen nil)
        (do ((G166451 pl (cdr G166451)) (G166425 nil))
            ((or (atom G166451)
                 (progn (setq G166425 (car G166451)) nil)
                 (progn
                  (progn
                   (setq prop (car G166425))
                   (setq val (cdr G166425))
                   G166425)
```

```
                        nil))
                    nil)
                (seq (exit
                 (cond
                  ((and (null (member prop propsSeen)) val)
                   (cond
                    ((|member| prop
                      '(|alias| |generatedCode| IS-GENSYM
                        |mapBody| |localVars|))
                     nil)
                    ((eq prop '|condition|)
                     (|displayCondition| prop val t))
                    ((eq prop '|recursive|)
                     (|sayMSG| "   This is recursive."))
                    ((eq prop '|isInterpreterFunction|)
                     (|sayMSG| "   This is an interpreter function.")
                     (|displayProperties,sayFunctionDeps| v))
                    ((eq prop '|isInterpreterRule|)
                     (|sayMSG| "   This is an interpreter rule.")
                     (|displayProperties,sayFunctionDeps| v))
                    ((eq prop '|localModemap|)
                     (|displayModemap| v val t))
                    ((eq prop '|mode|)
                     (|displayMode| prop val t))
                    (t
                     (when (eq prop '|value|)
                       (exit
                        (when val
                          (exit (|displayValue| v val t)))))
                     (|sayMSG| (list "   " prop ":  " val))
                     (setq propsSeen (cons prop propsSeen))))))))))))
          (t
           (|sayKeyedMsg| 'S2IZ0068 (cons option nil)))))))))
    (|terminateSystemCommand|)))))
```

———-

19.2.30 defun displayParserMacro

[pfPrintSrcLines p**??**]
[$pfMacros p98]

— defun displayParserMacro —

```
(defun |displayParserMacro| (m)
 (let ((m (assq m |$pfMacros|)))
 (declare (special |$pfMacros|))
```

```
  (when m (|pfPrintSrcLines| (caddr m)))))
```

———-

19.2.31 defun displayCondition

[bright p??]
[sayBrightly p??]
[concat p1041]
[pred2English p??]

— **defun displayCondition** —

```
(defun |displayCondition| (v condition giveVariableIfNil)
 (let (varPart condPart)
  (when giveVariableIfNil (setq varPart (cons '| of| (|bright| v))))
  (setq condPart (or condition '|true|))
  (|sayBrightly|
   (|concat| '|   condition| varPart '|:  | (|pred2English| condPart)))))
```

———-

19.2.32 defun interpFunctionDepAlists

[putalist p??]
[getalist p??]
[getFlag p??]
[$e p??]
[$dependeeAlist p??]
[$dependentAlist p??]
[$InteractiveFrame p??]

— **defun interpFunctionDepAlists** —

```
(defun |interpFunctionDepAlists| ()
 (let (|$e|)
 (declare (special |$e| |$dependeeAlist| |$dependentAlist|
                   |$InteractiveFrame|))
  (setq |$e| |$InteractiveFrame|)
  (setq |$dependentAlist| (cons (cons nil nil) nil))
  (setq |$dependeeAlist| (cons (cons nil nil) nil))
  (mapcar #'(lambda (dep)
   (let (dependee dependent)
    (setq dependee (first dep))
```

```
    (setq dependent (second dep))
    (setq |$dependentAlist|
     (putalist |$dependentAlist| dependee
      (cons dependent (getalist |$dependentAlist| dependee))))
    (spadlet |$dependeeAlist|
     (putalist |$dependeeAlist| dependent
      (cons dependee (getalist |$dependeeAlist| dependent))))))
   (|getFlag| '|$dependencies|))))
```

———-

19.2.33 defun displayModemap

[bright p??]
[sayBrightly p??]
[concat p1041]
[formatSignature p??]

— **defun displayModemap** —

```
(defun |displayModemap| (v val giveVariableIfNil)
 (labels (
  (g (v mm giveVariableIfNil)
   (let (local signature fn varPart prefix)
    (setq local (caar mm))
    (setq signature (cdar mm))
    (setq fn (cadr mm))
    (unless (eq local '|interpOnly|)
     (spadlet varPart (unless giveVariableIfNil (cons " of" (|bright| v))))
     (spadlet prefix
      (cons '|   Compiled function type| (append varPart (cons '|: | nil))))
     (|sayBrightly| (|concat| prefix (|formatSignature| signature)))))))
  (mapcar #'(lambda (x) (g v x giveVariableIfNil)) val)))
```

———-

19.2.34 defun displayMode

[bright p??]
[fixObjectForPrinting p452]
[sayBrightly p??]
[concat p1041]
[prefix2String p??]

— **defun displayMode** —

```
(defun |displayMode| (v mode giveVariableIfNil)
 (let (varPart)
  (when mode
   (unless giveVariableIfNil
    (setq varPart (cons '| of| (|bright| (|fixObjectForPrinting| v)))))
   (|sayBrightly|
    (|concat| '|   Declared type or mode| varPart '|:   |
     (|prefix2String| mode))))))
```

———-

19.2.35 defun Split into tokens delimted by spaces

[stripSpaces p467]

— **defun dumbTokenize** —

```
(defun |dumbTokenize| (str)
 (let (inString token (tokenStart 0) previousSpace tokenList)
  (dotimes (i (1- (|#| str)))
   (cond
    ((char= (elt str i) #\") ; don't split strings
     (setq inString (null inString))
     (setq previousSpace nil))
    ((and (char= (elt str i) #\space) (null inString))
     (unless previousSpace
       (setq token (|stripSpaces| (subseq str tokenStart i)))
       (setq tokenList (cons token tokenList))
       (setq tokenStart (1+ i))
       (setq previousSpace t)))
    (t
     (setq previousSpace nil))))
  (setq tokenList (cons (|stripSpaces| (subseq str tokenStart)) tokenList))
  (nreverse tokenList)))
```

———-

19.2.36 defun Convert string tokens to their proper type

[isIntegerString p464]

— **defun tokTran** —

```
(defun |tokTran| (tok)
 (let (tmp)
  (if (stringp tok)
   (cond
    ((eql (|#| tok) 0) nil)
    ((setq tmp (|isIntegerString| tok)) tmp)
    ((char= (elt tok 0) #\" ) (subseq tok 1 (1- (|#| tok))))
    (t (intern tok)))
   tok)))
```

———-

19.2.37 defun Is the argument string an integer?

— **defun isIntegerString 0** —

```
(defun |isIntegerString| (tok)
 (multiple-value-bind (int len) (parse-integer tok :junk-allowed t)
  (when (and int (= len (length tok))) int)))
```

———-

19.2.38 defun Handle parsed system commands

[dumbTokenize p463]
[parseSystemCmd p465]
[tokTran p463]
[systemCommand p444]

— **defun handleParsedSystemCommands** —

```
(defun |handleParsedSystemCommands| (unabr optionList)
 (declare (ignore unabr))
 (let (restOptionList parcmd trail)
  (setq restOptionList (mapcar #'|dumbTokenize| (cdr optionList)))
  (setq parcmd (|parseSystemCmd| (car optionList)))
  (setq trail
   (mapcar #'(lambda (opt)
               (mapcar #'(lambda (tok) (|tokTran| tok)) opt)) restOptionList))
  (|systemCommand| (cons parcmd trail)))))
```

———-

19.2.39 defun Parse a system command

[tokTran p463]
[stripSpaces p467]
[parseFromString p48]
[dumbTokenize p463]

— **defun parseSystemCmd** —

```
(defun |parseSystemCmd| (opt)
 (let (spaceIndex)
  (if (setq spaceIndex (search " " opt))
   (list
    (|tokTran| (|stripSpaces| (subseq opt 0 spaceIndex)))
    (|parseFromString| (|stripSpaces| (subseq opt spaceIndex))))
   (mapcar #'|tokTran| (|dumbTokenize| opt)))))
```

———-

19.2.40 defun Get first word in a string

[subseq p??]
[stringSpaces p??]

— **defun getFirstWord** —

```
(defun |getFirstWord| (string)
 (let (spaceIndex)
  (setq spaceIndex (search " " string))
  (if spaceIndex
   (|stripSpaces| (subseq string 0 spaceIndex))
   string)))
```

———-

19.2.41 defun Unabbreviate keywords in commands

[selectOptionLC p475]
[selectOption p475]
[commandsForUserLevel p444]
[$systemCommands p439]
[$currentLine p??]
[$syscommands p440]
[line p??]

— **defun unAbbreviateKeyword** —

```
(defun |unAbbreviateKeyword| (x)
 (let (xp)
 (declare (special |$systemCommands| |$currentLine| $syscommands line))
  (setq xp (|selectOptionLC| x $syscommands '|commandErrorIfAmbiguous|))
  (cond
   ((null xp)
    (setq xp '|system|)
    (setq line (concat ")system " (substring line 1 (1- (|#| line)))))
    (spadlet |$currentLine| line)))
  (|selectOption| xp (|commandsForUserLevel| |$systemCommands|)
     '|commandUserLevelError|)))
```

————-

19.2.42 defun The command is ambiguous error

[commandAmbiguityError p448]
[$oldline p445]
[line p??]

— **defun commandErrorIfAmbiguous** —

```
(defun |commandErrorIfAmbiguous| (x u)
 (declare (special $oldline line))
 (when u
   (setq $oldline line)
   (|commandAmbiguityError| '|command| x u)))
```

————-

[stripSpaces p467]
[nplisp p468]
[stripLisp p467]
[sayKeyedMsg p327]
[npboot p468]
[npsystem p468]
[npsynonym p469]
[member p1042]
[concat p1041]

— **defun handleNoParseCommands** —

```
(defun |handleNoParseCommands| (unab string)
```

```
(let (spaceindex funname)
 (setq string (|stripSpaces| string))
 (setq spaceindex (search " " string))
 (cond
  ((eq unab '|lisp|)
   (if spaceindex
    (|nplisp| (|stripLisp| string))
    (|sayKeyedMsg| 's2iv0005 nil)))
  ((eq unab '|boot|)
   (if spaceindex
    (|npboot| (subseq string (1+ spaceindex)))
    (|sayKeyedMsg| 's2iv0005 nil)))
  ((eq unab '|system|)
   (if spaceindex
     (|npsystem| unab string)
     (|sayKeyedMsg| 's2iv0005 nil)))
  ((eq unab '|synonym|)
   (if spaceindex
    (|npsynonym| unab (subseq string (1+ spaceindex)))
    (|npsynonym| unab "")))
  ((null spaceindex)
   (funcall unab))
  ((|member| unab '(|quit| |fin| |pquit| |credits| |copyright| |trademark|))
   (|sayKeyedMsg| 's2iv0005 nil))
  (t
   (setq funname (intern (concat "np" (string unab))))
   (funcall funname (subseq string (1+ spaceindex)))))))
```

———-

19.2.43 defun Remove the spaces surrounding a string

TPDHERE: This should probably be a macro or eliminated

— defun stripSpaces 0 —

```
(defun |stripSpaces| (str)
 (string-trim '(#\space) str))
```

———-

19.2.44 defun Remove the lisp command prefix

— defun stripLisp 0 —

```
(defun |stripLisp| (str)
```

```
 (if (string= (subseq str 0 4) "lisp")
  (subseq str 4)
  str))
```

———-

19.2.45 defun Handle the)lisp command

[$ans p**??**]

— **defun nplisp 0** —

```
(defun |nplisp| (str)
 (declare (special |$ans|))
  (setq |$ans| (eval (read-from-string str)))
  (format t "~&Value = ~S~%" |$ans|))
```

———-

19.2.46 defun The)boot command is no longer supported

TPDHERE: Remove all boot references from top level

— **defun npboot 0** —

```
(defun |npboot| (str)
 (declare (ignore str))
 (format t "The )boot command is no longer supported~%"))
```

———-

19.2.47 defun Handle the)system command

Note that unAbbreviateKeyword returns the word “system” for unknown words so we have to search for this case. This complication may never arrive in practice. [sayKeyedMsg p327]

— **defun npsystem** —

```
(defun |npsystem| (unab str)
 (let (spaceIndex sysPart)
  (setq spaceIndex (search " " str))
  (cond
   ((null spaceIndex) (|sayKeyedMsg| 'S2IZ0080 (list str)))
   (t
```

```
    (setq sysPart (subseq str 0 spaceIndex))
    (if (search sysPart (string unab))
     (obey (subseq str (1+ spaceIndex)))
     (|sayKeyedMsg| 'S2IZ0080 (list sysPart)))))))
```

———-

19.2.48 defun Handle the)synonym command

[npProcessSynonym p469]

— **defun npsynonym** —

```
(defun |npsynonym| (unab str)
 (declare (ignore unab))
 (|npProcessSynonym| str))
```

———-

19.2.49 defun Handle the synonym system command

[printSynonyms p470]
[processSynonymLine p827]
[putalist p**??**]
[terminateSystemCommand p448]
[$CommandSynonymAlist p474]

— **defun npProcessSynonym** —

```
(defun |npProcessSynonym| (str)
 (let (pair)
 (declare (special |$CommandSynonymAlist|))
  (if (= (length str) 0)
   (|printSynonyms| nil)
   (progn
    (setq pair (|processSynonymLine| str))
    (if |$CommandSynonymAlist|
     (putalist |$CommandSynonymAlist| (car pair) (cdr pair)))
     (setq |$CommandSynonymAlist| (cons pair nil))))
  (|terminateSystemCommand|)))
```

———-

19.2.50 defun printSynonyms

[centerAndHighlight p??]
[specialChar p970]
[filterListOfStringsWithFn p933]
[synonymsForUserLevel p825]
[printLabelledList p470]
[$CommandSynonymAlist p474]
[$linelength p765]

— **defun printSynonyms** —

```
(defun |printSynonyms| (patterns)
 (prog (ls t1)
  (declare (special |$CommandSynonymAlist| $linelength))
   (|centerAndHighlight| '|System Command Synonyms|
                 $linelength (|specialChar| '|hbar|))
   (setq ls
    (|filterListOfStringsWithFn| patterns
     (do ((t2 (|synonymsForUserLevel| |$CommandSynonymAlist|) (cdr t2)))
         ((atom t2) (nreverse0 t1))
        (push (cons (princ-to-string (caar t2)) (cdar t2)) t1))
     (|function| car)))
   (|printLabelledList| ls "user" "synonyms" ")" patterns)))
```

———-

19.2.51 defun Print a list of each matching synonym

The prefix goes before each element on each side of the list, eg, ")" [sayMessage p??]
[blankList p??]
[substring p??]
[entryWidth p??]
[sayBrightly p??]
[concat p1041]
[fillerSpaces p20]

— **defun printLabelledList** —

```
(defun |printLabelledList| (ls label1 label2 prefix patterns)
 (let (comm syn wid)
  (if (null ls)
   (if (null patterns)
    (|sayMessage| (list "   No " label1 "-defined " label2 " in effect."))
    (|sayMessage|
     `("   No " ,label1 "-defined " ,label2 " satisfying patterns:"
```

```
      |%l| "     " |%b| ,@(append (|blankList| patterns) (list '|%d|)))))
  (progn
   (when patterns
     (|sayMessage|
      `(,label1 "-defined " ,label2 " satisfying patterns:" |%l| "   "
         |%b| ,@(append (|blankList| patterns) (list '|%d|)))))
    (do ((t1 ls (cdr t1)))
        ((atom t1) nil)
     (setq syn (caar t1))
     (setq comm (cdar t1))
     (when (string= (substring syn 0 1) "|")
      (setq syn (substring syn 1 nil)))
     (when (string= syn "%i") (setq syn "%i "))
     (setq wid (max (- 30 (|entryWidth| syn)) 1))
     (|sayBrightly|
      (|concat| '|%b| prefix syn '|%d| (|fillerSpaces| wid ".")
        " " prefix comm)))
    (|sayBrightly| "")))))
```

———-

19.2.52 defvar $tokenCommands

This is a list of the commands that expect the interpreter to parse their arguments. Thus the history command expects that Axiom will have tokenized and validated the input before calling the history function.

— **initvars** —

```
(defvar |$tokenCommands| nil)
```

———-

— **postvars** —

```
(eval-when (eval load)
 (setq |$tokenCommands|
 '( |abbreviations|
    |cd|
    |clear|
    |close|
    |compiler|
    |depends|
    |display|
    |describe|
    |edit|
```

```
    |frame|
    |frame|
    |help|
    |history|
    |input|
    |library|
    |load|
    |ltrace|
    |read|
    |savesystem|
    |set|
    |spool|
    |undo|
    |what|
    |with|
    |workfiles|
    |zsystemdevelopment|
    )))
```

———-

19.2.53 defvar $InitialCommandSynonymAlist

Axiom can create "synonyms" for commands. We create an initial table of synonyms which are in common use.

— **initvars** —

```
(defvar |$InitialCommandSynonymAlist| nil)
```

———-

19.2.54 defun Print the current version information

[*yearweek* p**??**]
[*build-version* p**??**]

— **defun axiomVersion 0** —

```
(defun axiomVersion ()
 (declare (special *build-version* *yearweek*))
  (concatenate 'string "Axiom " *build-version* " built on " *yearweek*))
```

———-

— **postvars** —

```
(eval-when (eval load)
 (setq |$InitialCommandSynonymAlist|
   '(
       (|?|          . "what commands")
       (|ap|         . "what things")
       (|apr|        . "what things")
       (|apropos|    . "what things")
       (|cache|      . "set functions cache")
       (|cl|         . "clear")
       (|cls|        . "zsystemdevelopment )cls")
       (|cms|        . "system")
       (|co|         . "compiler")
       (|d|          . "display")
       (|dep|        . "display dependents")
       (|dependents| . "display dependents")
       (|e|          . "edit")
       (|expose|     . "set expose add constructor")
       (|fc|         . "zsystemdevelopment )c")
       (|fd|         . "zsystemdevelopment )d")
       (|fdt|        . "zsystemdevelopment )dt")
       (|fct|        . "zsystemdevelopment )ct")
       (|fctl|       . "zsystemdevelopment )ctl")
       (|fe|         . "zsystemdevelopment )e")
       (|fec|        . "zsystemdevelopment )ec")
       (|fect|       . "zsystemdevelopment )ect")
       (|fns|        . "exec spadfn")
       (|fortran|    . "set output fortran")
       (|h|          . "help")
       (|hd|         . "system hypertex &")
       (|kclam|      . "boot clearClams ( )")
       (|killcaches| . "boot clearConstructorAndLisplibCaches ( )")
       (|patch|      . "zsystemdevelopment )patch")
       (|pause|      . "zsystemdevelopment )pause")
       (|prompt|     . "set message prompt")
       (|recurrence| . "set functions recurrence")
       (|restore|    . "history )restore")
       (|save|       . "history )save")
       (|startGraphics|    .  "system $AXIOM/lib/viewman &")
       (|startNAGLink|     .  "system $AXIOM/lib/nagman &")
       (|stopGraphics|     .  "lisp (|sockSendSignal| 2 15)")
       (|stopNAGLink|      .  "lisp (|sockSendSignal| 8 15)")
       (|time|       . "set message time")
       (|type|       . "set message type")
       (|unexpose|   . "set expose drop constructor")
       (|up|         . "zsystemdevelopment )update")
       (|version|    . "lisp (axiomVersion)")
       (|w|          . "what")
```

```
       (|wc|         . "what categories")
       (|wd|         . "what domains")
       (|who|        . "lisp (pprint credits)")
       (|wp|         . "what packages")
       (|ws|         . "what synonyms")
)))
```

———-

19.2.55 defvar $CommandSynonymAlist

The actual list of synonyms is initialized to be the same as the above initial list of synonyms. The user synonyms that are added during a session are pushed onto this list for later lookup.

— **initvars** —

```
(defvar |$CommandSynonymAlist| nil)
```

———-

— **postvars** —

```
(eval-when (eval load)
 (setq |$CommandSynonymAlist| (copy-alist |$InitialCommandSynonymAlist|)))
```

———-

19.2.56 defun ncloopCommand

The $systemCommandFunction is set in SpadInterpretStream to point to the function InterpExecuteSpadSystemCommand. The system commands are handled by the function kept in the "hook" variable `$systemCommandFunction` which has the default function `InterpExecuteSpadSystemComma` Thus, when a system command is entered this function is called.

The only exception is the `)include` function which inserts the contents of a file inline in the input stream. This is useful for processing `)read` of input files. [ncloopPrefix? p475]
[ncloopInclude1 p617]
[$systemCommandFunction p**??**]
[$systemCommandFunction p**??**]

— **defun ncloopCommand** —

```
(defun |ncloopCommand| (line n)
 (let (a)
```

```
(declare (special |$systemCommandFunction|))
 (if (setq a (|ncloopPrefix?| ")include" line))
  (|ncloopInclude1| a n)
  (progn
   (funcall |$systemCommandFunction| line)
   n))))
```

———-

19.2.57 defun ncloopPrefix?

If we find the prefix string in the whole string starting at position zero we return the remainder of the string without the leading prefix.

— **defun ncloopPrefix? 0** —

```
(defun |ncloopPrefix?| (prefix whole)
 (when (eql (search prefix whole) 0)
  (subseq whole (length prefix))))
```

———-

19.2.58 defun selectOptionLC

[selectOption p475]
[downcase p**??**]
[object2Identifier p**??**]

— **defun selectOptionLC** —

```
(defun |selectOptionLC| (x l errorFunction)
 (|selectOption| (downcase (|object2Identifier| x)) l errorFunction))
```

———-

19.2.59 defun selectOption

[member p1042]
[identp p1040]
[stringPrefix? p**??**]
[pname p1039]
[qcdr p**??**]
[qcar p**??**]

— defun selectOption —

```
(defun |selectOption| (x l errorfunction)
 (let (u y)
  (cond
   ((|member| x l) x)
   ((null (identp x))
    (cond
     (errorfunction (funcall errorfunction x u))
     (t nil)))
   (t
    (setq u
     (let (t0)
       (do ((t1 l (cdr t1)) (y nil))
           ((or (atom t1) (progn (setq y (car t1)) nil)) (nreverse0 t0))
          (if (|stringPrefix?| (pname x) (pname y))
             (setq t0 (cons y t0))))))
    (cond
     ((and (consp u) (eq (qcdr u) nil) (progn (setq y (qcar u)) t)) y)
     (errorfunction (funcall errorfunction x u))
     (t nil))))))
```

———-

Chapter 20

)abbreviations help page Command

20.1 abbreviations help page man page

— abbreviations.help —

```
====================================================================
A.2.  )abbreviation
====================================================================

User Level Required:  compiler

Command Syntax:

  -  )abbreviation query  [nameOrAbbrev]
  -  )abbreviation category  abbrev  fullname []quiet]
  -  )abbreviation domain  abbrev  fullname   []quiet]
  -  )abbreviation package  abbrev  fullname  []quiet]
  -  )abbreviation remove  nameOrAbbrev

Command Description:

This command is used to query, set and remove abbreviations for category,
domain and package constructors. Every constructor must have a unique
abbreviation. This abbreviation is part of the name of the subdirectory under
which the components of the compiled constructor are stored. Furthermore, by
issuing this command you let the system know what file to load automatically
if you use a new constructor. Abbreviations must start with a letter and then
be followed by up to seven letters or digits. Any letters appearing in the
abbreviation must be in uppercase.
```

```
When used with the query argument, this command may be used to list the name
associated with a particular abbreviation or the abbreviation for a
constructor. If no abbreviation or name is given, the names and corresponding
abbreviations for all constructors are listed.

The following shows the abbreviation for the constructor List:

)abbreviation query List

The following shows the constructor name corresponding to the abbreviation
NNI:

)abbreviation query NNI

The following lists all constructor names and their abbreviations.

)abbreviation query

To add an abbreviation for a constructor, use this command with category,
domain or package. The following add abbreviations to the system for a
category, domain and package, respectively:

)abbreviation domain   SET Set
)abbreviation category COMPCAT  ComplexCategory
)abbreviation package  LIST2MAP ListToMap

If the )quiet option is used, no output is displayed from this command. You
would normally only define an abbreviation in a library source file. If this
command is issued for a constructor that has already been loaded, the
constructor will be reloaded next time it is referenced. In particular, you
can use this command to force the automatic reloading of constructors.

To remove an abbreviation, the remove argument is used. This is usually only
used to correct a previous command that set an abbreviation for a constructor
name. If, in fact, the abbreviation does exist, you are prompted for
confirmation of the removal request. Either of the following commands will
remove the abbreviation VECTOR2 and the constructor name VectorFunctions2
from the system:

)abbreviation remove VECTOR2
)abbreviation remove VectorFunctions2

Also See:
o )compile
```

———-

20.2 Functions

20.2.1 defun abbreviations

[abbreviationsSpad2Cmd p479]

— **defun abbreviations** —

```
(defun |abbreviations| (l)
 (|abbreviationsSpad2Cmd| l))
```

————-

20.2.2 defun abbreviationsSpad2Cmd

[listConstructorAbbreviations p480]
[abbreviation? p**??**]
[abbQuery p532]
[deldatabase p1001]
[size p1039]
[sayKeyedMsg p327]
[mkUserConstructorAbbreviation p**??**]
[setdatabase p1000]
[seq p**??**]
[exit p**??**]
[opOf p**??**]
[helpSpad2Cmd p568]
[selectOptionLC p475]
[qcar p**??**]
[qcdr p**??**]
[$options p**??**]

— **defun abbreviationsSpad2Cmd** —

```
(defun |abbreviationsSpad2Cmd| (arg)
 (let (abopts quiet opt key type constructor t2 a b al)
  (declare (special |$options|))
  (if (null arg)
   (|helpSpad2Cmd| '(|abbreviations|))
   (progn
    (setq abopts '(|query| |domain| |category| |package| |remove|))
    (setq quiet nil)
    (do ((t0 |$options| (cdr t0)) (t1 nil))
        ((or (atom t0)
             (progn (setq t1 (car t0)) nil)
```

```
          (progn (progn (setq opt (car t1)) t1) nil))
        nil)
   (setq opt (|selectOptionLC| opt '(|quiet|) '|optionError|))
   (when (eq opt '|quiet|) (setq quiet t)))
 (when
  (and (consp arg)
       (progn
        (setq opt (qcar arg))
        (setq al (qcdr arg))
        t))
   (setq key (|opOf| (car al)))
   (setq type (|selectOptionLC| opt abopts '|optionError|))
   (cond
    ((eq type '|query|)
     (cond
      ((null al) (|listConstructorAbbreviations|))
      ((setq constructor (|abbreviation?| key))
        (|abbQuery| constructor))
      (t (|abbQuery| key))))
    ((eq type '|remove|)
     (deldatabase key 'abbreviation))
    ((oddp (size al))
     (|sayKeyedMsg| 's2iz0002 (list type)))
    (t
     (do () (nil nil)
      (seq
       (exit
        (cond
         ((null al) (return '|fromLoop|))
         (t
           (setq t2 al)
           (setq a (car t2))
           (setq b (cadr t2))
           (setq al (cddr t2))
           (|mkUserConstructorAbbreviation| b a type)
           (setdatabase b 'abbreviation a)
           (setdatabase b 'constructorkind type))))))
     (unless quiet
       (|sayKeyedMsg| 's2iz0001 (list a type (|opOf| b))))))))))
```

———-

20.2.3 defun listConstructorAbbreviations

[upcase p**??**]
[queryUserKeyedMsg p**??**]
[string2id-n p**??**]
[whatSpad2Cmd p930]

[sayKeyedMsg p327]

— **defun listConstructorAbbreviations** —

```
(defun |listConstructorAbbreviations| ()
 (let (x)
  (setq x (upcase (|queryUserKeyedMsg| 's2iz0056 nil)))
  (if (member (string2id-n x 1) '(Y YES))
   (progn
    (|whatSpad2Cmd| '(|categories|))
    (|whatSpad2Cmd| '(|domains|))
    (|whatSpad2Cmd| '(|packages|)))
   (|sayKeyedMsg| 's2iz0057 nil))))
```

————-

Chapter 21

)boot help page Command

21.1 boot help page man page

— boot.help —

```
=======================================================================
A.3.  )boot
=======================================================================

User Level Required:  development

Command Syntax:

  -  )boot bootExpression

Command Description:

This command is used by AXIOM system developers to execute expressions
written in the BOOT language. For example,

)boot times3(x) == 3*x

creates and compiles the Lisp function ‘‘times3’’ obtained by translating the
BOOT code.

Also See:
o )fin
o )lisp
o )set
o )system
```

———-

1

21.2 Functions

This command is in the list of `$noParseCommands` 19.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 19.2.1

[1] "fin" (32.1.1 p 544) "lisp" (?? p ??) "set" (45.36.1 p 799) "system" (?? p ??)

Chapter 22

)browse help page Command

22.1 browse help page man page

— **browse.help** —

```
User Level Required: development

Command Syntax:

  )browse

Command Description:

This command is used by Axiom system users to start the Axiom top level
loop listening for browser connections.
```

———-

22.2 Overview

The Axiom book on the help browser is a complete rewrite of the hyperdoc mechanism. There are several components that were needed to make this function. Most of the web browser components are described in bookvol11.pamphlet. This portion describes some of the design issues needed to support the interface.

The axServer command takes a port (defaulting to 8085) and a program to handle the browser interaction (defaulting to multiServ). The axServer function opens the port, constructs the

stream, and passes the stream to multiServ. The multiServ loop processes one interaction at a time.

So the basic process is that the Axiom ")browse" command opens a socket and listens for http requests. Based on the type of request (either 'GET' or 'POST') and the content of the request, which is one of:

- command - algebra request/response
- lispcall - a lisp s-expression to be evaluated
- showcall - an Axiom)show command

the multiServ function will call a handler function to evaluate the command line and construct a response. GET requests result in a new browser page. POST requests result in an inline result.

Most responses contain the fields:

- stepnum - this is the Axiom step number
- command - this is the original command from the browser
- algebra - this is the Axiom 2D algebra output
- mathml - this is the MathML version of the Axiom algebra
- type - this is the type of the Axiom result

22.3 Browsers, MathML, and Fonts

This work has the Firefox browser as its target. Firefox has built-in support for MathML, javascript, and XMLHttpRequest handling. More details are available in bookvol11.pamphlet but the very basic machinery for communication with the browser involves a dance between the browser and the multiServ function (see the axserver.spad.pamphlet).

In particular, a simple request is embedded in a web page as:

```
<ul>
 <li>
  <input type="submit" id="p3" class="subbut"
    onclick="makeRequest('p3');"
    value="sin(x)" />
  <div id="ansp3"><div></div></div>
 </li>
</ul>
```

which says that this is an html "input" field of type "submit". The CSS display class is "subbut" which is of a different color than the surrounding text to make it obvious that you can click on this field. Clickable fields that have no response text are of class "noresult".

The javascript call to "makeRequest" gives the "id" of this input field, which must be unique in the page, as an argument. In this case, the argument is 'p3'. The "value" field holds the display text which will be passed back to Axiom as a command.

When the result arrives the "showanswer" function will select out the mathml field of the response, construct the "id" of the html div to hold the response by concatenating the string "ans" (answer) to the "id" of the request resulting, in this case, as "ansp3". The "showanswer" function will find this div and replace it with a div containing the mathml result.

The "makeRequest" function is:

```
function makeRequest(arg) {
  http_request = new XMLHttpRequest();
  var command = commandline(arg);
  //alert(command);
  http_request.open('POST', '127.0.0.1:8085', true);
  http_request.onreadystatechange = handleResponse;
  http_request.setRequestHeader('Content-Type', 'text/plain');
  http_request.send("command="+command);
  return(false);
```

It contains a request to open a local server connection to Axiom, sets "handleResponse" as the function to call on reply, sets up the type of request, fills in the command field, and sends off the http request.

When a response is received, the "handleResponse" function checks for the correct reply state, strips out the important text, and calls "showanswer".

```
function handleResponse() {
 if (http_request.readyState == 4) {
  if (http_request.status == 200) {
   showanswer(http_request.responseText,'mathAns');
  } else
  {
    alert('There was a problem with the request.'+ http_request.statusText);
  }
 }
}
```

See bookvol11.pamphlet for further details.

22.4 The axServer/multiServ loop

The basic call to start an Axiom browser listener is:

```
)set message autoload off
)set output mathml on
axServer(8085,multiServ)$AXSERV
```

This call sets the port, opens a socket, attaches it to a stream, and then calls "multiServ" with that stream. The "multiServ" function loops serving web responses to that port.

22.5 The)browse command

In order to make the whole process cleaner the function ")browse" handles the details. This code creates the command-line function for)browse

The browse function does the internal equivalent of the following 3 command line statments:

```
)set message autoload off
)set output mathml on
axServer(8085,multiServ)$AXSERV
```

which causes Axiom to start serving web pages on port 8085

For those unfamiliar with calling algebra from lisp there are a few points to mention.

The loadLib needs to be called to load the algebra code into the image. Normally this is automatic but we are not using the interpreter so we need to do this "by hand".

Each algebra file contains a "constructor function" which builds the domain, which is a vector, and then caches the vector so that every call to the contructor returns an EQ vector, that is, the same vector. In this case, we call the constructor |AxiomServer|

The axServer function was mangled internally to |AXSERV;axServer;IMV;2|. The multiServ function was mangled to |AXSERV;multiServ;SeV;3| Note well that if you change axserver.spad these names might change which will generate the error message along the lines of:

```
System error:
The function $\vert$AXSERV;axServer;IMV;2$\vert$ is undefined.
```

To fix this you need to look at int/algebra/AXSERV.nrlib/code.lsp and find the new mangled function name. A better solution would be to dynamically look up the surface names in the domain vector.

Each Axiom function expects the domain vector as the last argument. This is not obvious from the call as the interpreter supplies it. We must do that "by hand".

We don't call the multiServ function. We pass it as a parameter to the axServer function. When it does get called by the SPADCALL macro it needs to be a lisp pair whose car is the function and whose cdr is the domain vector. We construct that pair here as the second argument to axServer. The third, hidden, argument to axServer is the domain vector which we supply "by hand".

The socket can be supplied on the command line but defaults to 8085. Axiom supplies the arguments as a list.

22.6 Variables Used

22.7 Functions

[set p799]
[loadLib p1029]
[AxiomServer p**??**]
[AXSERV;axServer;IMV;2 p**??**]

— defun browse —

```
(defun |browse| (socket)
 (let (axserv browser)
  (if socket
    (setq socket (car socket))
    (setq socket 8085))
  (|set| '(|mes| |auto| |off|))
  (|set| '(|out| |mathml| |on|))
  (|loadLib| '|AxiomServer|)
  (setq axserv (|AxiomServer|))
  (setq browser
   (|AXSERV;axServer;IMV;2| socket
    (cons #'|AXSERV;multiServ;SeV;3| axserv) axserv))))
```

————-

Now we have to bolt it into Axiom. This involves two lookups.

We create the lisp pair

```
  (|browse| . |development|)
```

and cons it into the $systemCommands command table. This allows the command to be executed in development mode. This lookup decides if this command is allowed. It also has the side-effect of putting the command into the $SYSCOMMANDS variable which is used to determine if the token is a command.

22.8 The server support code

Chapter 23

)cd help page Command

23.1 cd help page man page

— cd.help —

```
========================================================================
A.4.  )cd
========================================================================

User Level Required:  interpreter

Command Syntax:

  -  )cd directory

Command Description:

This command sets the AXIOM working current directory. The current directory
is used for looking for input files (for )read), AXIOM library source files
(for )compile), saved history environment files (for )history )restore),
compiled AXIOM library files (for )library), and files to edit (for )edit).
It is also used for writing spool files (via )spool), writing history input
files (via )history )write) and history environment files (via )history
)save),and compiled AXIOM library files (via )compile).

If issued with no argument, this command sets the AXIOM current directory to
your home directory. If an argument is used, it must be a valid directory
name. Except for the ``)'' at the beginning of the command, this has the same
syntax as the operating system cd command.

Also See:
o )compile
```

o)edit
o)history
o)library
o)read
o)spool

———-

1

23.2 Variables Used

23.3 Functions

[1] "edit" (31.2.1 p 540) "history" (35.4.7 p 578) "library" (65.1.34 p 1005) "read" (43.1.1 p 638) "spool" (?? p ??)

Chapter 24

)clear help page Command

24.1 clear help page man page

— clear.help —

```
====================================================================
A.6.  )clear
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )clear all
  - )clear completely
  - )clear properties all
  - )clear properties  obj1 [obj2 ...]
  - )clear value      all
  - )clear value     obj1 [obj2 ...]
  - )clear mode       all
  - )clear mode      obj1 [obj2 ...]

Command Description:

This command is used to remove function and variable declarations,
definitions and values from the workspace. To empty the entire workspace and
reset the step counter to 1, issue

)clear all

To remove everything in the workspace but not reset the step counter, issue
```

```
)clear properties all

To remove everything about the object x, issue

)clear properties x

To remove everything about the objects x, y and f, issue

)clear properties x y f

The word properties may be abbreviated to the single letter ‘‘p’’.

)clear p all
)clear p x
)clear p x y f

All definitions of functions and values of variables may be removed by either

)clear value all
)clear v all

This retains whatever declarations the objects had. To remove definitions and
values for the specific objects x, y and f, issue

)clear value x y f
)clear v x y f

To remove the declarations of everything while leaving the definitions and
values, issue

)clear mode  all
)clear m all

To remove declarations for the specific objects x, y and f, issue

)clear mode x y f
)clear m x y f

The )display names and )display properties commands may be used to see what
is currently in the workspace.

The command

)clear completely

does everything that )clear all does, and also clears the internal system
function and constructor caches.

Also See:
o )display
```

```
o )history
o )undo
```

———-

[1]

24.2 Variables Used

24.2.1 defvar $clearOptions

— **initvars** —

```
(defvar |$clearOptions| '(|modes| |operations| |properties| |types| |values|))
```

———-

24.3 Functions

24.3.1 defun clear

[clearSpad2Cmd p496]

— **defun clear** —

```
(defun |clear| (l)
 (|clearSpad2Cmd| l))
```

———-

24.3.2 defvar $clearExcept

— **initvars** —

```
(defvar |$clearExcept| nil)
```

———-

[1] "display" (30.2.1 p 531) "history" (35.4.7 p 578) "undo" (52.4.6 p 912)

24.3.3 defun clearSpad2Cmd

TPDHERE: Note that this function also seems to parse out)except)completely and)scaches which don't seem to be documented. [selectOptionLC p475]
[sayKeyedMsg p327]
[clearCmdAll p499]
[clearCmdCompletely p498]
[clearCmdSortedCaches p497]
[clearCmdExcept p500]
[clearCmdParts p501]
[updateCurrentInterpreterFrame p555]
[$clearExcept p495]
[$options p**??**]
[$clearOptions p495]

— defun clearSpad2Cmd —

```
(defun |clearSpad2Cmd| (l)
  (let (|$clearExcept| opt optlist arg)
  (declare (special |$clearExcept| |$options| |$clearOptions|))
  (cond
   (|$options|
    (setq |$clearExcept|
     (prog (t0)
      (setq t0 t)
      (return
       (do ((t1 nil (null t0))
            (t2 |$options| (cdr t2))
            (t3 nil))
           ((or t1
                (atom t2)
                (progn (setq t3 (car t2)) nil)
                (progn (progn (setq opt (car t3)) t3) nil))
             t0)
        (setq t0
         (and t0
              (eq
               (|selectOptionLC| opt '(|except|) '|optionError|)
               '|except|)))))))))
  (cond
   ((null l)
     (setq optlist
      (prog (t4)
       (setq t4 nil)
       (return
        (do ((t5 |$clearOptions| (cdr t5)) (x nil))
            ((or (atom t5) (progn (setq x (car t5)) nil)) t4)
         (setq t4 (append t4 `(|%l| "       " ,x)))))))
     (|sayKeyedMsg| 's2iz0010 (list optlist)))
```

```
   (t
     (setq arg
      (|selectOptionLC| (car l) '(|all| |completely| |scaches|) nil))
     (cond
      ((eq arg '|all|)        (|clearCmdAll|))
      ((eq arg '|completely|) (|clearCmdCompletely|))
      ((eq arg '|scaches|)    (|clearCmdSortedCaches|))
      (|$clearExcept|         (|clearCmdExcept| l))
      (t
       (|clearCmdParts| l)
       (|updateCurrentInterpreterFrame|)))))))
```

———-

24.3.4 defun clearCmdSortedCaches

[compiledLookupCheck p497]
[spadcall p??]
[$lookupDefaults p??]
[$Void p??]
[$ConstructorCache p??]

— **defun clearCmdSortedCaches** —

```
(defun |clearCmdSortedCaches| ()
 (let (|$lookupDefaults| domain pair)
  (declare (special |$lookupDefaults| |$Void| |$ConstructorCache|))
  (do ((t0 (hget |$ConstructorCache| '|SortedCache|) (cdr t0))
       (t1 nil))
      ((or (atom t0)
           (progn
            (setq t1 (car t0))
            (setq domain (cddr t1))
            nil))
        nil)
    (setq pair (|compiledLookupCheck| '|clearCache| (list |$Void|) domain))
    (spadcall pair))))
```

———-

24.3.5 defun compiledLookupCheck

[compiledLookup p1061]
[keyedSystemError p??]
[formatSignature p??]

— **defun compiledLookupCheck** —

```
(defun |compiledLookupCheck| (op sig dollar)
 (let (fn)
  (setq fn (|compiledLookup| op sig dollar))
  (cond
    ((and (null fn) (eq op '^))
      (setq fn (|compiledLookup| '** sig dollar)))
    ((and (null fn) (eq op '**))
      (setq fn (|compiledLookup| '^ sig dollar)))
    (t nil))
  (cond
   ((null fn)
    (|keyedSystemError| 'S2NR0001
      (list op (|formatSignature| sig) (elt dollar 0))))
   (t fn))))
```

———-

24.3.6 defvar $functionTable

— **initvars** —

```
(defvar |$functionTable| nil)
```

———-

24.3.7 defun clearCmdCompletely

[clearCmdAll p499]
 [sayKeyedMsg p327]
 [clearClams p**??**]
 [clearConstructorCaches p**??**]
 [reclaim p39]
 [$localExposureData p688]
 [$xdatabase p**??**]
 [$CatOfCatDatabase p**??**]
 [$DomOfCatDatabase p**??**]
 [$JoinOfCatDatabase p**??**]
 [$JoinOfDomDatabase p**??**]
 [$attributeDb p**??**]
 [$functionTable p498]

[$existingFiles p??]
[$localExposureDataDefault p688]

— **defun clearCmdCompletely** —

```
(defun |clearCmdCompletely| ()
  (declare (special |$localExposureData| |$xdatabase| |$CatOfCatDatabase|
    |$DomOfCatDatabase| |$JoinOfCatDatabase| |$JoinOfDomDatabase|
    |$attributeDb| |$functionTable| |$existingFiles|
    |$localExposureDataDefault|))
  (|clearCmdAll|)
  (setq |$localExposureData| (copy-seq |$localExposureDataDefault|))
  (setq |$xdatabase| nil)
  (setq |$CatOfCatDatabase| nil)
  (setq |$DomOfCatDatabase| nil)
  (setq |$JoinOfCatDatabase| nil)
  (setq |$JoinOfDomDatabase| nil)
  (setq |$attributeDb| nil)
  (setq |$functionTable| nil)
  (|sayKeyedMsg| 's2iz0013 nil)
  (|clearClams|)
  (|clearConstructorCaches|)
  (setq |$existingFiles| (make-hash-table :test #'equal))
  (|sayKeyedMsg| 's2iz0014 nil)
  (reclaim)
  (|sayKeyedMsg| 's2iz0015 nil))
```

———-

24.3.8 defun clearCmdAll

[clearCmdSortedCaches p497]
[untraceMapSubNames p863]
[resetInCoreHist p584]
[deleteFile p1036]
[histFileName p576]
[updateCurrentInterpreterFrame p555]
[clearMacroTable p500]
[sayKeyedMsg p327]
[$frameRecord p911]
[$previousBindings p911]
[$variableNumberAlist p??]
[$InteractiveFrame p??]
[$useInternalHistoryTable p575]
[$internalHistoryTable p??]
[$frameMessages p732]

[$interpreterFrameName p**??**]
[$currentLine p**??**]

— **defun clearCmdAll** —

```
(defun |clearCmdAll| ()
  (declare (special |$frameRecord| |$previousBindings| |$variableNumberAlist|
     |$InteractiveFrame| |$useInternalHistoryTable| |$internalHistoryTable|
     |$frameMessages| |$interpreterFrameName| |$currentLine|))
  (|clearCmdSortedCaches|)
  (setq |$frameRecord| nil)
  (setq |$previousBindings| nil)
  (setq |$variableNumberAlist| nil)
  (|untraceMapSubNames| /tracenames)
  (setq |$InteractiveFrame| (list (list nil)))
  (|resetInCoreHist|)
  (when |$useInternalHistoryTable|
    (setq |$internalHistoryTable| nil)
   (|deleteFile| (|histFileName|)))
  (setq |$IOindex| 1)
  (|updateCurrentInterpreterFrame|)
  (setq |$currentLine| ")clear all")
  (|clearMacroTable|)
  (when |$frameMessages|
    (|sayKeyedMsg| 's2iz0011 (list |$interpreterFrameName|))
    (|sayKeyedMsg| 's2iz0012 nil)))
```

———-

24.3.9 defun clearMacroTable

[$pfMacros p98]

— **defun clearMacroTable 0** —

```
(defun |clearMacroTable| ()
  (declare (special |$pfMacros|))
  (setq |$pfMacros| nil))
```

———-

24.3.10 defun clearCmdExcept

Clear all the options except the argument. [stringPrefix? p**??**]
[object2String p**??**]

[clearCmdParts p501]
[$clearOptions p495]

— **defun clearCmdExcept** —

```
(defun |clearCmdExcept| (arg)
 (let ((opt (car arg)) (vl (cdr arg)))
 (declare (special |$clearOptions|))
  (dolist (option |$clearOptions|)
   (unless (|stringPrefix?| (|object2String| opt) (|object2String| option))
    (|clearCmdParts| (cons option vl))))))
```

———-

24.3.11 defun clearCmdParts

[selectOptionLC p475]
[pname p1039]
[types p??]
[modes p??]
[values p??]
[boot-equal p??]
[assocleft p??]
[remdup p??]
[assoc p??]
[isMap p??]
[get p??]
[exit p??]
[untraceMapSubNames p863]
[seq p??]
[recordOldValue p588]
[recordNewValue p587]
[deleteAssoc p??]
[sayKeyedMsg p327]
[getParserMacroNames p449]
[getInterpMacroNames p??]
[clearDependencies p??]
[member p1042]
[clearParserMacro p449]
[sayMessage p??]
[fixObjectForPrinting p452]
[$e p??]
[$InteractiveFrame p??]
[$clearOptions p495]

— defun clearCmdParts —

```
(defun |clearCmdParts| (arg)
 (let (|$e| (opt (car arg)) option pmacs imacs (vl (cdr arg)) p1 lm prop p2)
 (declare (special |$e| |$InteractiveFrame| |$clearOptions|))
  (setq option (|selectOptionLC| opt |$clearOptions| '|optionError|))
  (setq option (intern (pname option)))
  (setq option
   (case option
    (|types| '|mode|)
    (|modes| '|mode|)
    (|values| '|value|)
    (t option)))
  (if (null vl)
   (|sayKeyedMsg| 's2iz0055 nil)
   (progn
    (setq pmacs (|getParserMacroNames|))
    (setq imacs (|getInterpMacroNames|))
    (cond
     ((boot-equal vl '(|all|))
      (setq vl (assocleft (caar |$InteractiveFrame|)))
      (setq vl (remdup (append vl pmacs)))))
    (setq |$e| |$InteractiveFrame|)
    (do ((t0 vl (cdr t0)) (x nil))
        ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
      (|clearDependencies| x t)
      (when (and (eq option '|properties|) (|member| x pmacs))
        (|clearParserMacro| x))
      (when (and (eq option '|properties|)
                 (|member| x imacs)
                 (null (|member| x pmacs)))
        (|sayMessage| (cons
         "   You cannot clear the definition of the system-defined macro "
          (cons (|fixObjectForPrinting| x)
                (cons (intern "." "BOOT") nil)))))
      (cond
       ((setq p1 (|assoc| x (caar |$InteractiveFrame|)))
        (cond
         ((eq option '|properties|)
          (cond
           ((|isMap| x)
            (seq
             (cond
              ((setq lm
                (|get| x '|localModemap| |$InteractiveFrame|))
               (cond
                ((consp lm)
                 (exit (|untraceMapSubNames| (cons (cadar lm) nil))))))
              (t nil)))))
          (dolist (p2 (cdr p1))
```

```
          (setq prop (car p2))
          (|recordOldValue| x prop (cdr p2))
          (|recordNewValue| x prop nil))
        (setf (caar |$InteractiveFrame|)
         (|deleteAssoc| x (caar |$InteractiveFrame|))))
       ((setq p2 (|assoc| option (cdr p1)))
        (|recordOldValue| x option (cdr p2))
        (|recordNewValue| x option nil)
        (rplacd p2 nil))))))
nil))))
```

———-

Chapter 25

)close help page Command

25.1 close help page man page

— close.help —

```
=======================================================================
A.5.  )close
=======================================================================

User Level Required:  interpreter

Command Syntax:

  - )close
  - )close )quietly

Command Description:

This command is used to close down interpreter client processes. Such
processes are started by HyperDoc to run AXIOM examples when you click on
their text. When you have finished examining or modifying the example and you
do not want the extra window around anymore, issue

)close

to the AXIOM prompt in the window.

If you try to close down the last remaining interpreter client process, AXIOM
will offer to close down the entire AXIOM session and return you to the
operating system by displaying something like

   This is the last AXIOM session. Do you want to kill AXIOM?
```

```
Type "y" (followed by the Return key) if this is what you had in mind. Type
"n" (followed by the Return key) to cancel the command.

You can use the )quietly option to force AXIOM to close down the interpreter
client process without closing down the entire AXIOM session.

Also See:
o )quit
o )pquit
```

———-

[1]

25.2 Functions

25.2.1 defun queryClients

Returns the number of active scratchpad clients [sockSendInt p**??**]
[sockGetInt p**??**]
[$SessionManager p**??**]
[$QueryClients p**??**]

— **defun queryClients** —

```
(defun |queryClients| ()
  (declare (special |$SessionManager| |$QueryClients|))
  (|sockSendInt| |$SessionManager| |$QueryClients|)
  (|sockGetInt| |$SessionManager|))
```

———-

25.2.2 defun close

[throwKeyedMsg p**??**]
[sockSendInt p**??**]
[closeInterpreterFrame p558]
[selectOptionLC p475]
[upcase p**??**]
[queryUserKeyedMsg p**??**]
[string2id-n p**??**]

[1] "quit" (42.2.1 p 634) "pquit" (41.2.1 p 630)

[queryClients p506]
[$SpadServer p12]
[$SessionManager p**??**]
[$CloseClient p**??**]
[$currentFrameNum p43]
[$options p**??**]

— **defun close** —

```
(defun |close| (args)
 (declare (ignore args))
 (let (numClients opt fullopt quiet x)
 (declare (special |$SpadServer| |$SessionManager| |$CloseClient|
     |$currentFrameNum| |$options|))
  (if (null |$SpadServer|)
   (|throwKeyedMsg| 's2iz0071 nil))
   (progn
    (setq numClients (|queryClients|))
    (cond
     ((> numClients 1)
       (|sockSendInt| |$SessionManager| |$CloseClient|)
       (|sockSendInt| |$SessionManager| |$currentFrameNum|)
       (|closeInterpreterFrame| nil))
     (t
      (do ((t0 |$options| (cdr t0)) (t1 nil))
          ((or (atom t0)
               (progn (setq t1 (car t0)) nil)
               (progn (progn (setq opt (car t1)) t1) nil))
              nil)
       (setq fullopt (|selectOptionLC| opt '(|quiet|) '|optionError|))
       (unless quiet (setq quiet (eq fullopt '|quiet|))))
      (cond
       (quiet
        (|sockSendInt| |$SessionManager| |$CloseClient|)
        (|sockSendInt| |$SessionManager| |$currentFrameNum|)
        (|closeInterpreterFrame| nil))
       (t
        (setq x (upcase (|queryUserKeyedMsg| 's2iz0072 nil)))
        (when (member (string2id-n x 1) '(yes y)) (bye)))))))))
```

————-

Chapter 26

)compile help page Command

26.1 compile help page man page

— compile.help —

```
====================================================================
A.7.  )compile
====================================================================

User Level Required:  compiler

Command Syntax:

  -  )compile
  -  )compile fileName
  -  )compile fileName.spad
  -  )compile directory/fileName.spad
  -  )compile fileName )quiet
  -  )compile fileName )noquiet
  -  )compile fileName )break
  -  )compile fileName )nobreak
  -  )compile fileName )library
  -  )compile fileName )nolibrary
  -  )compile fileName )vartrace
  -  )compile fileName )constructor nameOrAbbrev

Command Description:

You use this command to invoke the AXIOM library compiler.  This
compiles files with file extension .spad with the AXIOM system
compiler. The command first looks in the standard system directories
for files with extension .spad.
```

Should you not want the)library command automatically invoked, call)compile with the)nolibrary option. For example,

```
)compile mycode )nolibrary
```

By default, the)library system command exposes all domains and categories it processes. This means that the AXIOM intepreter will consider those domains and categories when it is trying to resolve a reference to a function. Sometimes domains and categories should not be exposed. For example, a domain may just be used privately by another domain and may not be meant for top-level use. The)library command should still be used, though, so that the code will be loaded on demand. In this case, you should use the)nolibrary option on)compile and the)noexpose option in the)library command. For example,

```
)compile mycode.spad )nolibrary
)library mycode )noexpose
```

Once you have established your own collection of compiled code, you may find it handy to use the)dir option on the)library command. This causes)library to process all compiled code in the specified directory. For example,

```
)library )dir /u/jones/as/quantum
```

You must give an explicit directory after)dir, even if you want all compiled code in the current working directory processed.

```
)library )dir .
```

You can compile category, domain, and package constructors contained in files with file extension .spad. You can compile individual constructors or every constructor in a file.

The full filename is remembered between invocations of this command and)edit commands. The sequence of commands

```
)compile matrix.spad
)edit
)compile
```

will call the compiler, edit, and then call the compiler again on the file matrix.spad. If you do not specify a directory, the working current directory (see description of command)cd) is searched for the file. If the file is not found, the standard system directories are searched.

If you do not give any options, all constructors within a file are compiled. Each constructor should have an)abbreviation command in the file in which it is defined. We suggest that you place the)abbreviation commands at the top of the file in the order in which the constructors are defined. The list of

```
commands serves as a table of contents for the file.

The )library option causes directories containing the compiled code for each
constructor to be created in the working current directory. The name of such
a directory consists of the constructor abbreviation and the .NRLIB file
extension. For example, the directory containing the compiled code for the
MATRIX constructor is called MATRIX.NRLIB. The )nolibrary option says that
such files should not be created.

The )vartrace option causes the compiler to generate extra code for the
constructor to support conditional tracing of variable assignments. (see
description of command )trace ). Without this option, this code is suppressed
and one cannot use the )vars option for the trace command.

The )constructor option is used to specify a particular constructor to
compile. All other constructors in the file are ignored. The constructor name
or abbreviation follows )constructor. Thus either

)compile matrix.spad )constructor RectangularMatrix

or

)compile matrix.spad )constructor RMATRIX

compiles the RectangularMatrix constructor defined in matrix.spad.

The )break and )nobreak options determine what the compiler does
when it encounters an error. )break is the default and it indicates that
processing should stop at the first error. The value of the )set break
variable then controls what happens.

Also See:
o )abbreviation
o )edit
o )library
```

———-

1

26.2 Functions

26.2.1 defvar $/editfile

— initvars —

[1] “abbreviation” (?? p ??) “edit” (31.2.1 p 540) “library” (65.1.34 p 1005)

```
(defvar /editfile nil)
```

———-

Chapter 27

)copyright help page Command

27.1 copyright help page man page

— copyright.help —

```
The term Axiom, in the field of computer algebra software,
along with AXIOM and associated images are common-law
trademarks. While the software license allows copies, the
trademarks may only be used when referring to this project.

Axiom is distributed under terms of the Modified BSD license.
Axiom was released under this license as of September 3, 2002.
Source code is freely available at:
http://savannah.nongnu.org/projects/axiom
Copyrights remain with the original copyright holders.
Use of this material is by permission and/or license.
Individual files contain reference to these applicable copyrights.
The copyright and license statements are collected here for reference.

Portions Copyright (c) 2003- The Axiom Team

The Axiom Team is the collective name for the people who have
contributed to this project. Where no other copyright statement
is noted in a file this copyright will apply.

Portions Copyright (c) 1991-2002, The Numerical ALgorithms Group Ltd.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:
```

- Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in
  the documentation and/or other materials provided with the
  distribution.

- Neither the name of The Numerical ALgorithms Group Ltd. nor the
  names of its contributors may be used to endorse or promote products
  derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions Copyright (C) 1989-95 GROUPE BULL

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to
deal in the Software without restriction, including without limitation the
rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
GROUPE BULL BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of GROUPE BULL shall not be
used in advertising or otherwise to promote the sale, use or other dealings
in this Software without prior written authorization from GROUPE BULL.

Portions Copyright (C) 2002, Codemist Ltd. All rights reserved.
acn@codemist.co.uk

CCL Public License 1.0
==========

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

(1) Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

(2) Redistributions in binary form must reproduce the above copyright
notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

(3) Neither the name of Codemist nor the names of other contributors may
be used to endorse or promote products derived from this software without
specific prior written permission.

(4) If you distribute a modified form or either source or binary code
  (a) you must make the source form of these modification available
      to Codemist;
  (b) you grant Codemist a royalty-free license to use, modify
      or redistribute your modifications without limitation;
  (c) you represent that you are legally entitled to grant these rights
      and that you are not providing Codemist with any code that violates
      any law or breaches any contract.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Portions Copyright (C) 1995-1997 Eric Young (eay@mincom.oz.au)
All rights reserved.

This package is an SSL implementation written
by Eric Young (eay@mincom.oz.au).
The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as
the following conditions are aheared to. The following conditions

apply to all code found in this distribution, be it the RC4, RSA,
lhash, DES, etc., code; not just the SSL code. The SSL documentation
included with this distribution is covered by the same copyright terms
except that the holder is Tim Hudson (tjh@mincom.oz.au).

Copyright remains Eric Young's, and as such any Copyright notices in
the code are not to be removed.
If this package is used in a product, Eric Young should be given attribution
as the author of the parts of the library used.
This can be in the form of a textual message at program startup or
in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software
   must display the following acknowledgement:
   "This product includes cryptographic software written by
    Eric Young (eay@mincom.oz.au)"
   The word 'cryptographic' can be left out if the rouines from the library
   being used are not cryptographic related :-).
4. If you include any Windows specific code (or a derivative thereof) from
   the apps directory (application code) you must include an acknowledgement:
   "This product includes software written by Tim Hudson (tjh@mincom.oz.au)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

The licence and distribution terms for any publically available version or
derivative of this code cannot be changed. i.e. this code cannot simply be
copied and put under another distribution licence
[including the GNU Public Licence.]

Portions Copyright (C) 1988 by Leslie Lamport.

Portions Copyright (c) 1998 Free Software Foundation, Inc.

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, distribute with modifications, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE ABOVE COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR
THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name(s) of the above copyright
holders shall not be used in advertising or otherwise to promote the
sale, use or other dealings in this Software without prior written
authorization.

Portions Copyright 1989-2000 by Norman Ramsey. All rights reserved.

Noweb is protected by copyright. It is not public-domain
software or shareware, and it is not protected by a ‘‘copyleft’’
agreement like the one used by the Free Software Foundation.

Noweb is available free for any use in any field of endeavor. You may
redistribute noweb in whole or in part provided you acknowledge its
source and include this COPYRIGHT file. You may modify noweb and
create derived works, provided you retain this copyright notice, but
the result may not be called noweb without my written consent.

You may sell noweb if you wish. For example, you may sell a CD-ROM
including noweb.

You may sell a derived work, provided that all source code for your
derived work is available, at no additional charge, to anyone who buys
your derived work in any form. You must give permisson for said
source code to be used and modified under the terms of this license.
You must state clearly that your work uses or is based on noweb and
that noweb is available free of change. You must also request that
bug reports on your work be reported to you.

Portions Copyright (c) 1987 The RAND Corporation. All rights reserved.

Portions Copyright 1988-1995 by Stichting Mathematisch Centrum, Amsterdam, The Netherlands.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Portions Copyright (c) Renaud Rioboo and the University Paris 6.

Portions Copyright (c) 2003-2010 Jocelyn Guidry

Portions Copyright (c) 2001-2010 Timothy Daly

———-

27.2 Functions

27.2.1 defun copyright

[obey p**??**]
[concat p1041]
[getenviron p31]

— **defun copyright** —

```
(defun |copyright| ()
 (obey (concat "cat " (getenviron "AXIOM") "/doc/spadhelp/copyright.help")))
```

———-

27.2.2 defun trademark

— **defun trademark 0** —

```
(defun |trademark| ()
 (format t "The term Axiom, in the field of computer algebra software, ~%")
 (format t "along with AXIOM and associated images are common-law ~%")
 (format t "trademarks. While the software license allows copies, the ~%")
 (format t "trademarks may only be used when referring to this project ~%"))
```

————-

This command is in the list of `$noParseCommands` 19.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 19.2.1

Chapter 28

)credits help page Command

28.1 credits help page man page

28.2 Variables Used

28.3 Functions

28.3.1 defun credits

[credits p521]

— **defun credits 0** —

```
(defun |credits| ()
 (declare (special credits))
 (mapcar #'(lambda (x) (princ x) (terpri)) credits))
```

————-

This command is in the list of `$noParseCommands` 19.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 19.2.1

Chapter 29

)describe help page Command

29.1 describe help page man page

— describe.help —

```
====================================================================
  )describe
====================================================================

User Level Required:  interpreter

Command Syntax:

  -  )describe categoryName
  -  )describe domainName
  -  )describe packageName

Command Description:

This command is used to display the comments for the operation, category,
domain or package. The comments are part of the algebra source code.

The commands

)describe <categoryName> [internal]
)describe <domainName>   [internal]
)describe <packageName>  [internal]

will show a properly formatted version of the "Description:" keyword
from the comments in the algebra source for the category, domain,
or package requested.
```

```
If 'internal' is requested, then the internal format of the domain or
package is described. Categories do not have an internal representation.
```

———-

29.1.1 defvar $describeOptions

The current value of $describeOptions is

— **initvars** —

```
(defvar $describeOptions '(|category| |domain| |package|))
```

———-

29.2 Functions

29.2.1 defun Print comment strings from algebra libraries

This trivial function satisfies the standard pattern of making a user command match the name of the function which implements the command. That command immediatly invokes a "Spad2Cmd" version. [describespad2cmd p**??**]

— **defun describe** —

```
(defun |describe| (l)
 (describeSpad2Cmd l))
```

———-

29.2.2 defun describeSpad2Cmd

The describe command prints cleaned-up comment strings from the algebra libraries. It can print strings associated with a category, domain, package, or by operation.

This implements command line options of the form:

```
  )describe categoryName [internal]
  )describe domainName   [internal]
  )describe packageName  [internal]
```

The describeInternal function will either call the "dc" function to describe the internal representation of the argument or it will print a cleaned up version of the text for the

"Description" keyword in the Category, Domain, or Package source code. [selectOptionLC p475]
[flatten p527]
[cleanline p525]
[getdatabase p1001]
[sayMessage p**??**]
[$e p**??**]
[$EmptyEnvironment p**??**]
[$describeOptions p524]

— **defun describeSpad2Cmd** —

```
(defun describeSpad2Cmd (l)
 (labels (
  (fullname (arg)
   "Convert abbreviations to the full constructor name"
   (let ((abb (getdatabase arg 'abbreviation)))
    (if abb arg (getdatabase arg 'constructor))))
  (describeInternal (cdp internal?)
   (if internal?
    (progn
     (unless (eq (getdatabase cdp 'constructorkind) '|category|) (|dc| cdp))
     (showdatabase cdp))
    (mapcar #'(lambda (x) (if (stringp x) (cleanline x)))
     (flatten  (car (getdatabase (fullname cdp) 'documentation)))))))
 (let ((|$e| |$EmptyEnvironment|) (opt (second l)))
  (declare (special |$e| |$EmptyEnvironment| $describeOptions))
  (if (and (consp l) (not (eq opt '?)))
    (describeInternal (first l) (second l))
   (|sayMessage|
    (append
     '("  )describe keyword arguments are")
     (mapcar #'(lambda (x) (format nil "~%     ~a" x)) $describeOptions)
     (format nil "~%  or abbreviations thereof")))))))
```

———-

29.2.3 defun cleanline

— **defun cleanline** —

```
(defun cleanline (line)
 (labels (
   (replaceInLine (thing other line)
    (do ((mark (search thing line) (search thing line)))
        ((null mark) line)
```

```
    (setq line
     (concatenate 'string (subseq line 0 mark) other
                          (subseq line (+ mark (length thing)))))))

  (removeFromLine (thing line) (replaceInLine thing "" line))

  (removeKeyword (str line)
   (do ((mark (search str line) (search str line)))
    ((null mark) line)
    (let (left point mid right)
     (setq left (subseq line 0 mark))
     (setq point (search "}" line :start2 mark))
     (setq mid (subseq line (+ mark (length str)) point))
     (setq right (subseq line (+ point 1)))
     (setq line (concatenate 'string left mid right)))))

  (addSpaces (str line)
   (do ((mark (search str line) (search str line)) (cnt))
    ((null mark) line)
    (let (left point mid right)
     (setq left (subseq line 0 mark))
     (setq point (search "}" line :start2 mark))
     (setq mid (subseq line (+ mark (length str)) point))
     (if (setq cnt (parse-integer mid :junk-allowed t))
       (setq mid (make-string cnt :initial-element #\ ))
       (setq mid ""))
     (setq right (subseq line (+ point 1)))
     (setq line (concatenate 'string left mid right)))))

  (splitAtNewline (line)
   (do ((mark (search "~%" line) (search "~%" line)) (lines))
       ((null mark)
         (push " " lines)
         (push line lines)
         (nreverse lines))
     (push (subseq line 0 mark) lines)
     (setq line (subseq line (+ mark 2)))))

  (wrapOneLine (line margin result)
   (if (null line)
     (nreverse result)
     (if (< (length line) margin)
       (wrapOneLine nil margin (append (list line) result))
       (let (oneline spill aspace)
        (setq aspace (position #\space (subseq line 0 margin) :from-end t))
        (setq oneline (string-trim '(#\space) (subseq line 0 aspace)))
        (setq spill (string-trim '(#\space) (subseq line aspace)))
        (wrapOneLine spill margin (append (list oneline) result))))))

  (reflowParagraph (line)
```

```
    (let (lst1)
     (setq lst1 (splitAtNewLine line))
     (dolist (x lst1)
      (mapcar #'(lambda(y) (format t "~a~%" y))
       (wrapOneLine x 70 nil))))))

 (setq line (removeFromLine "{}" line))
 (setq line (replaceInLine "\\blankline" "~%~%" line))
 (setq line (replaceInLine "\\br" "~%" line))
 (setq line (removeFromLine "\\" line))
 (dolist (str '("spad{" "spadtype{" "spadop{" "spadfun{" "spadatt{"
                "axiom{" "axiomType{" "spadignore{" "axiomFun{"
                "centerline{" "inputbitmap{" "axiomOp{" "spadgloss{"))
  (setq line (removeKeyword str line)))
 (setq line (replaceInLine "{e.g.}" "e.g." line))
 (dolist (str '("tab{" "indented{" ))
   (setq line (addSpaces str line)))
  (reflowParagraph line)))
```

———-

29.2.4 defun flatten

— **defun flatten 0** —

```
(defun flatten (x)
 (labels (
  (rec (x acc)
   (cond
    ((null x) acc)
    ((atom x) (cons x acc))
    (t (rec (car x) (rec (cdr x) acc))))))
  (rec x nil)))
```

———-

Chapter 30

)display help page Command

30.1 display help page man page

— display.help —

```
=======================================================================
A.8.  )display
=======================================================================

User Level Required:  interpreter

Command Syntax:

  -  )display all
  -  )display properties
  -  )display properties all
  -  )display properties [obj1 [obj2 ...] ]
  -  )display value all
  -  )display value [obj1 [obj2 ...] ]
  -  )display mode all
  -  )display mode [obj1 [obj2 ...] ]
  -  )display names
  -  )display operations opName

Command Description:

This command is used to display the contents of the workspace and signatures
of functions with a given name. (A signature gives the argument and return
types of a function.)

The command
```

```
)display names

lists the names of all user-defined objects in the workspace. This is useful
if you do not wish to see everything about the objects and need only be
reminded of their names.

The commands

)display all
)display properties
)display properties all

all do the same thing: show the values and types and declared modes of all
variables in the workspace. If you have defined functions, their signatures
and definitions will also be displayed.

To show all information about a particular variable or user functions, for
example, something named d, issue

)display properties d

To just show the value (and the type) of d, issue

)display value d

To just show the declared mode of d, issue

)display mode d

All modemaps for a given operation may be displayed by using )display
operations. A modemap is a collection of information about a particular
reference to an operation. This includes the types of the arguments and the
return value, the location of the implementation and any conditions on the
types. The modemap may contain patterns. The following displays the modemaps
for the operation FromcomplexComplexCategory:

)d op complex

Also See:
o )clear
o )history
o )set
o )show
o )what
```

1

[1] "clear" (24.3.1 p 495) "history" (35.4.7 p 578) "set" (45.36.1 p 799) "show" (46.1.1 p 806) "what"

30.1.1 defvar $displayOptions

The current value of $displayOptions is

— **initvars** —

```
(defvar |$displayOptions|
 '(|abbreviations| |all| |macros| |modes| |names| |operations|
   |properties| |types| |values|))
```

————-

30.2 Functions

30.2.1 defun display

This trivial function satisfies the standard pattern of making a user command match the name of the function which implements the command. That command immediatly invokes a "Spad2Cmd" version. [displayspad2cmd p**??**]

— **defun display** —

```
(defun |display| (l)
 (displaySpad2Cmd l))
```

————-

30.2.2 displaySpad2Cmd

We process the options to the command and call the appropriate display function. There are really only 4 display functions. All of the other options are just subcases.

There is a slight mismatch between the $displayOptions list of symbols and the options this command accepts so we have a cond branch to clean up the option variable. This allows for the options to be plural.

If we fall all the way thru we use the $displayOptions list to construct a list of strings for the sayMessage function and tell the user what options are available. [abbQuery p532]
[opOf p**??**]
[listConstructorAbbreviations p480]
[displayOperations p533]
[displayMacros p534]
[displayWorkspaceNames p450]

(53.1.2 p 929)

[displayProperties p457]
[selectOptionLC p475]
[sayMessage p??]
[$e p??]
[$EmptyEnvironment p??]
[$displayOptions p531]

— **defun displaySpad2Cmd** —

```
(defun displaySpad2Cmd (l)
 (let ((|$e| |$EmptyEnvironment|) (opt (car l)) (vl (cdr l)) option)
  (declare (special |$e| |$EmptyEnvironment| |$displayOptions|))
  (if (and (consp l) (not (eq opt '?)))
   (progn
    (setq option (|selectOptionLC| opt |$displayOptions| '|optionError|))
    (cond
     ((eq option '|all|)
       (setq l (list '|properties|))
       (setq option '|properties|))
     ((or (eq option '|modes|) (eq option '|types|))
       (setq l (cons '|type| vl))
       (setq option '|type|))
     ((eq option '|values|)
       (setq l (cons '|value| vl))
       (setq option '|value|)))
    (cond
     ((eq option '|abbreviations|)
      (if (null vl)
        (|listConstructorAbbreviations|)
        (dolist (v vl) (|abbQuery| (|opOf| v)))))
     ((eq option '|operations|) (|displayOperations| vl))
     ((eq option '|macros|) (|displayMacros| vl))
     ((eq option '|names|) (|displayWorkspaceNames|))
     (t (|displayProperties| option l))))
   (|sayMessage|
    (append
      '("  )display keyword arguments are")
      (mapcar #'(lambda (x) (format nil "~%     ~a" x)) |$displayOptions|)
      (format nil "~%  or abbreviations thereof"))))))
```

———-

30.2.3 defun abbQuery

[getdatabase p1001]
[sayKeyedMsg p327]

— **defun abbQuery** —

```
(defun |abbQuery| (x)
 (let (abb)
  (cond
   ((setq abb (getdatabase x 'abbreviation))
    (|sayKeyedMsg| 's2iz0001 (list abb (getdatabase x 'constructorkind) x)))
   ((setq abb (getdatabase x 'constructor))
    (|sayKeyedMsg| 's2iz0001 (list x (getdatabase abb 'constructorkind) abb)))
   (t
     (|sayKeyedMsg| 's2iz0003 (list x))))))
```

———-

30.2.4 defun displayOperations

This function takes a list of operation names. If the list is null we query the user to see if they want all operations printed. Otherwise we print the information for the requested symbols. [reportOpSymbol p**??**]
[yesanswer p533]
[sayKeyedMsg p327]

— **defun displayOperations** —

```
(defun |displayOperations| (l)
 (if l
  (dolist (op l) (|reportOpSymbol| op))
  (if (yesanswer)
   (dolist (op (|allOperations|)) (|reportOpSymbol| op))
   (|sayKeyedMsg| 's2iz0059 nil))))
```

———-

30.2.5 defun yesanswer

This is a trivial function to simplify the logic of displaySpad2Cmd. If the user didn't supply an argument to the)display op command we ask if they wish to have all information about all Axiom operations displayed. If the answer is either Y or YES we return true else nil. [string2id-n p**??**]
[upcase p**??**]
[queryUserKeyedMsg p**??**]

— **defun yesanswer** —

```
(defun yesanswer ()
 (member
  (string2id-n (upcase (|queryUserKeyedMsg| 's2iz0058 nil)) 1) '(y yes)))
```

———-

30.2.6 defun displayMacros

[getInterpMacroNames p**??**]
[getParserMacroNames p449]
[remdup p**??**]
[sayBrightly p**??**]
[member p1042]
[displayParserMacro p460]
[seq p**??**]
[exit p**??**]
[displayMacro p449]

— **defun displayMacros** —

```
(defun |displayMacros| (names)
 (let (imacs pmacs macros first)
  (setq imacs (|getInterpMacroNames|))
  (setq pmacs (|getParserMacroNames|))
  (if names
    (setq macros names)
    (setq macros (append imacs pmacs)))
  (setq macros (remdup macros))
  (cond
   ((null macros) (|sayBrightly| "   There are no Axiom macros."))
   (t
     (setq first t)
     (do ((t0 macros (cdr t0)) (macro nil))
         ((or (atom t0) (progn (setq macro (car t0)) nil)) nil)
      (seq
       (exit
        (cond
         ((|member| macro pmacs)
           (cond
            (first (|sayBrightly|
             (cons '|%l| (cons "User-defined macros:" nil))) (setq first nil)))
           (|displayParserMacro| macro))
         ((|member| macro imacs) '|iterate|)
         (t (|sayBrightly|
          (cons "   "
           (cons '|%b|
            (cons macro
```

```
        (cons '|%d| (cons " is not a known Axiom macro." nil))))))))))))
(setq first t)
(do ((t1 macros (cdr t1)) (macro nil))
    ((or (atom t1) (progn (setq macro (car t1)) nil)) nil)
 (seq
  (exit
   (cond
    ((|member| macro imacs)
      (cond
       ((|member| macro pmacs) '|iterate|)
       (t
        (cond
         (first
          (|sayBrightly|
           (cons '|%l|
            (cons "System-defined macros:" nil))) (setq first nil)))
        (|displayMacro| macro))))
    ((|member| macro pmacs) '|iterate|)))))
nil))))
```

———-

30.2.7 defun sayExample

This function expects 2 arguments, the documentation string and the name of the operation. It searches the documentation string for `++X` lines. These lines are examples lines for functions. They look like ordinary `++` comments and fit into the ordinary comment blocks. So, for example, in the plot.spad.pamphlet file we find the following function signature:

```
plot: (F -> F,R) -> %
  ++ plot(f,a..b) plots the function \spad{f(x)}
  ++ on the interval \spad{[a,b]}.
  ++
  ++X fp:=(t:DFLOAT):DFLOAT +-> sin(t)
  ++X plot(fp,-1.0..1.0)$PLOT
```

This function splits out and prints the lines that begin with `++X`.

A minor complication of printing the examples is that the lines have been processed into internal compiler format. Thus the lines that read:

```
++X fp:=(t:DFLOAT):DFLOAT +-> sin(t)
++X plot(fp,-1.0..1.0)$PLOT
```

are actually stored as one long line containing the example lines

```
"\\indented{1}{plot(\\spad{f},{}a..\\spad{b}) plots the function
```

```
\\spad{f(x)}} \\indented{1}{on the interval \\spad{[a,{}b]}.}
\\blankline
\\spad{X} fp:=(t:DFLOAT):DFLOAT +-> sin(\\spad{t})
\\spad{X} plot(\\spad{fp},{}\\spad{-1}.0..1.0)\\$PLOT"
```

So when we have an example line starting with ++X, it gets converted to the compiler to `\spad{X}`. So each example line is delimited by `\spad{X}`.

The compiler also removes the newlines so if there is a subsequent `\spad{X}` in the docstring then it implies multiple example lines and we loop over them, splitting them up at the delimiter.

If there is only one then we clean it up and print it. [cleanupline p**??**]
[sayNewLine p**??**]

— defun sayExample —

```
(defun sayExample (docstring)
 (let (line point)
  (when (setq point (search "spad{X}" docstring))
   (setq line (subseq docstring (+ point 8)))
   (do ((mark (search "spad{X}" line) (search "spad{X}" line)))
     ((null mark))
    (princ (cleanupLine (subseq line 0 mark)))
    (|sayNewLine|)
    (setq line (subseq line (+ mark 8))))
   (princ (cleanupLine line))
   (|sayNewLine|)
   (|sayNewLine|))))
```

———-

30.2.8 defun cleanupLine

This function expects example lines in internal format that has been partially processed to remove the prefix. Thus we get lines that look like:

```
fp:=(t:DFLOAT):DFLOAT +-> sin(\\spad{t})
plot(\\spad{fp},{}\\spad{-1}.0..1.0)\\$PLOT
```

It removes all instances of `{}`, and \, and unwraps the `spad{}` call, leaving only the argument.

We return lines that look like:

```
fp:=(t:DFLOAT):DFLOAT +-> sin(t)
plot(fp,-1.0..1.0)$PLOT
```

which is hopefully exactly what the user wrote.

The compiler inserts `{}` as a space so we remove it. We remove all of the `\` characters. We remove all of the `spad{...}` delimiters which will occur around other spad variables. Technically we should search recursively for the matching delimiter rather than the next brace but the problem does not arise in practice.

— defun cleanupLine 0 —

```
(defun cleanupLine (line)
 (do ((mark (search "{}" line) (search "{}" line)))
  ((null mark))
  (setq line
   (concatenate 'string (subseq line 0 mark) (subseq line (+ mark 2)))))
 (do ((mark (search "\\" line) (search "\\" line)))
  ((null mark))
  (setq line
   (concatenate 'string (subseq line 0 mark) (subseq line (+ mark 1)))))
 (do ((mark (search "spad{" line) (search "spad{" line)))
  ((null mark))
  (let (left point mid right)
   (setq left (subseq line 0 mark))
   (setq point (search "}" line :start2 mark))
   (setq mid (subseq line (+ mark 5) point))
   (setq right (subseq line (+ point 1)))
   (setq line (concatenate 'string left mid right))))
 line)
```

———-

Chapter 31

)edit help page Command

31.1 edit help page man page

— edit.help —

```
========================================================================
A.9.  )edit
========================================================================

User Level Required:  interpreter

Command Syntax:

  - )edit [filename]

Command Description:

This command is used to edit files. It works in conjunction with the )read
and )compile commands to remember the name of the file on which you are
working. By specifying the name fully, you can edit any file you wish. Thus

)edit /u/julius/matrix.input

will place you in an editor looking at the file /u/julius/matrix.input. By
default, the editor is vi, but if you have an EDITOR shell environment
variable defined, that editor will be used. When AXIOM is running under the X
Window System, it will try to open a separate xterm running your editor if it
thinks one is necessary. For example, under the Korn shell, if you issue

export EDITOR=emacs

then the emacs editor will be used by )edit.
```

```
If you do not specify a file name, the last file you edited, read or compiled
will be used. If there is no ‘‘last file’’ you will be placed in the editor
editing an empty unnamed file.

It is possible to use the )system command to edit a file directly. For
example,

)system emacs /etc/rc.tcpip

calls emacs to edit the file.

Also See:
o )system
o )compile
o )read
```

———-

[1]

31.2 Functions

31.2.1 defun edit

[editSpad2Cmd p540]

— **defun edit** —

```
(defun |edit| (l) (|editSpad2Cmd| l))
```

———-

31.2.2 defun editSpad2Cmd

[pathname p1036]
[pathnameDirectory p1035]
[pathnameType p1034]
[$FINDFILE p**??**]
[pathnameName p1034]
[editFile p541]
[updateSourceFiles p542]
[/editfile p511]

[1] “system” (**??** p **??**) “read” (43.1.1 p 638)

— **defun editSpad2Cmd** —

```
(defun |editSpad2Cmd| (l)
 (let (olddir filetypes ll rc)
 (declare (special /editfile))
    (setq l (cond ((null l) /editfile) (t (car l))))
    (setq l (|pathname| l))
    (setq olddir (|pathnameDirectory| l))
    (setq filetypes
     (cond
      ((|pathnameType| l) (list (|pathnameType| l)))
      ((eq |$UserLevel| '|interpreter|) '("input" "INPUT" "spad" "SPAD"))
      ((eq |$UserLevel| '|compiler|) '("input" "INPUT" "spad" "SPAD"))
      (t '("input" "INPUT" "spad" "SPAD" "boot" "BOOT"
           "lisp" "LISP" "meta" "META"))))
    (setq ll
     (cond
      ((string= olddir "")
        (|pathname| ($findfile (|pathnameName| l) filetypes)))
      (t l)))
    (setq l (|pathname| ll))
    (setq /editfile l)
    (setq rc (|editFile| l))
    (|updateSourceFiles| l)
    rc))
```

———-

31.2.3 defun Implement the)edit command

[strconc p??]
[namestring p1034]
[pathname p1036]
[obey p??]

— **defun editFile** —

```
(defun |editFile| (file)
 (cond
  ((member (intern "WIN32" (find-package 'keyword)) *features*)
   (obey (strconc "notepad " (|namestring| (|pathname| file)))))
  (t
   (obey
    (strconc "$AXIOM/lib/SPADEDIT " (|namestring| (|pathname| file)))))))
```

———-

31.2.4 defun updateSourceFiles

[pathname p1036]
[pathnameName p1034]
[pathnameType p1034]
[makeInputFilename p973]
[member p1042]
[pathnameTypeId p1035]
[insert p**??**]
[$sourceFiles p**??**]

— **defun updateSourceFiles** —

```
(defun |updateSourceFiles| (arg)
 (declare (special |$sourceFiles|))
  (setq arg (|pathname| arg))
  (setq arg (|pathname| (list (|pathnameName| arg) (|pathnameType| arg) "*")))
  (when (and (makeInputFilename arg)
             (|member| (|pathnameTypeId| arg) '(boot lisp meta)))
   (setq |$sourceFiles| (|insert| arg |$sourceFiles|)))
  arg)
```

————-

Chapter 32

)fin help page Command

32.1 fin help page man page

— fin.help —

```
========================================================================
A.10.  )fin
========================================================================

User Level Required:  development

Command Syntax:

  -  )fin

Command Description:

This command is used by AXIOM developers to leave the AXIOM system and return
to the underlying Lisp system. To return to AXIOM, issue the ‘‘(spad)’’
function call to Lisp.

Also See:
o )pquit
o )quit
```

———-

1

[1] “pquit” (41.2.1 p 630) “quit” (42.2.1 p 634)

32.1.1 defun Exit from the interpreter to lisp

[spad-reader p??]
[eof p??]

— **defun fin 0** —

```
(defun |fin| ()
 (setq *eof* t)
 (throw 'spad_reader nil))
```

————-

32.2 Functions

This command is in the list of `$noParseCommands` 19.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 19.2.1

Chapter 33

)frame help page Command

33.1 frame help page man page

— frame.help —

```
====================================================================
A.11.  )frame
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )frame  new  frameName
  - )frame  drop  [frameName]
  - )frame  next
  - )frame  last
  - )frame  names
  - )frame  import frameName [objectName1 [objectName2 ...] ]
  - )set message frame on | off
  - )set message prompt frame

Command Description:

A frame can be thought of as a logical session within the physical session
that you get when you start the system. You can have as many frames as you
want, within the limits of your computer's storage, paging space, and so on.
Each frame has its own step number, environment and history. You can have a
variable named a in one frame and it will have nothing to do with anything
that might be called a in any other frame.

Some frames are created by the HyperDoc program and these can have pretty
```

```
strange names, since they are generated automatically. To find out the names
of all frames, issue

)frame names

It will indicate the name of the current frame.

You create a new frame ‘‘quark’’ by issuing

)frame new quark

The history facility can be turned on by issuing either )set history on or
)history )on. If the history facility is on and you are saving history
information in a file rather than in the AXIOM environment then a history
file with filename quark.axh will be created as you enter commands. If you
wish to go back to what you were doing in the ‘‘initial’’ frame, use

)frame next

or

)frame last

to cycle through the ring of available frames to get back to ‘‘initial’’.

If you want to throw away a frame (say ‘‘quark’’), issue

)frame drop quark

If you omit the name, the current frame is dropped.

If you do use frames with the history facility on and writing to a file, you
may want to delete some of the older history files. These are directories, so
you may want to issue a command like rm -r quark.axh to the operating system.

You can bring things from another frame by using )frame import. For example,
to bring the f and g from the frame ‘‘quark’’ to the current frame, issue

)frame import quark f g

If you want everything from the frame ‘‘quark’’, issue

)frame import quark

You will be asked to verify that you really want everything.

There are two )set flags to make it easier to tell where you are.

)set message frame on | off
```

```
will print more messages about frames when it is set on. By default, it is
off.

)set message prompt frame

will give a prompt that looks like

initial (1) ->

when you start up. In this case, the frame name and step make up the prompt.

Also See:
o )history
o )set
```

---

[1]

33.2 Variables Used

The frame mechanism uses several dollar variables.

33.2.1 Primary variables

Primary variables are those which exist solely to make the frame mechanism work.

The $interpreterFrameName contains a symbol which is the name of the current frame in use.

The $interpreterFrameRing contains a list of all of the existing frames. The first frame on the list is the "current" frame. When AXIOMsys is started directly there is only one frame named "initial".

If the system is started under sman (using the axiom shell script, for example), there are two frames, "initial" and "frame0". In this case, "frame0" is the current frame. This can cause subtle problems because functions defined in the axiom initialization file (.axiom.input) will be defined in frame "initial" but the current frame will be "frame0". They will appear to be undefined. However, if the user does ")frame next" they can switch to the "initial" frame and see the functions correctly defined.

The $frameMessages variable controls when frame messages will be displayed. The variable is initially NIL. It can be set on (T) or off (NIL) using the system command:

```
)set message frame on | off
```

[1] "history" (35.4.7 p 578) "set" (45.36.1 p 799)

Setting frame messages on will output a line detailing the current frame after every output is complete.

33.2.2 Used variables

The frame collects and uses a few top level variables. These are: $InteractiveFrame, $IOindex, $HiFiAccess, $HistList, $HistListLen, $HistListAct, $HistRecord, $internalHistoryTable, and $localExposureData.

These variables can also be changed by the frame mechanism when the user requests changing to a different frame.

33.3 Data Structures

33.3.1 Frames and the Interpreter Frame Ring

Axiom has the notion of "frames". A frame is a data structure which holds all the vital data from an Axiom session. There can be multiple frames and these live in a top-level variable called $interpreterFrameRing. This variable holds a circular list of frames. The parts of a frame and their initial, default values are:

```
$interpreterFrameName      a string, named on creation
$InteractiveFrame          (list (list nil))
$IOindex                   an integer, 1
$HiFiAccess                $HiFiAccess, see the variable description
$HistList                  $HistList, see the variable description
$HistListLen               $HistListLen, see the variable description
$HistListAct               $HistListAct, see the variable description
$HistRecord                $HistRecord, see the variable description
$internalHistoryTable      nil
$localExposureData         a copy of $localExposureData
```

33.4 Accessor Functions

These could be macros but we wish to export them to the API code in the algebra so we keep them as functions.

33.4.1 0th Frame Component – frameName

33.4.2 defun frameName

— defun frameName 0 —

```
(defun frameName (frame)
 (car frame))
```

———-

33.4.3 1st Frame Component – frameInteractive

— **defun frameInteractive 0** —

```
(defun frameInteractive (frame)
 (nth 1 frame))
```

———-

33.4.4 2nd Frame Component – frameIOIndex

— **defun frameIOIndex 0** —

```
(defun frameIOIndex (frame)
 (nth 2 frame))
```

———-

33.4.5 3rd Frame Component – frameHiFiAccess

— **defun frameHiFiAccess 0** —

```
(defun frameHiFiAccess (frame)
 (nth 3 frame))
```

———-

33.4.6 4th Frame Component – frameHistList

— **defun frameHistList 0** —

```
(defun frameHistList (frame)
 (nth 4 frame))
```

———-

33.4.7 5th Frame Component – frameHistListLen

— defun frameHistListLen 0 —

```
(defun frameHistListLen (frame)
 (nth 5 frame))
```

———-

33.4.8 6th Frame Component – frameHistListAct

— defun frameHistListAct 0 —

```
(defun frameHistListAct (frame)
 (nth 6 frame))
```

———-

33.4.9 7th Frame Component – frameHistRecord

— defun frameHistRecord 0 —

```
(defun frameHistRecord (frame)
 (nth 7 frame))
```

———-

33.4.10 8th Frame Component – frameHistoryTable

— defun frameHistoryTable 0 —

```
(defun frameHistoryTable (frame)
 (nth 8 frame))
```

———-

33.4.11 9th Frame Component – frameExposureData

— **defun frameExposureData 0** —

```
(defun frameExposureData (frame)
 (nth 9 frame))
```

———-

33.5 Functions

33.5.1 Initializing the Interpreter Frame Ring

Now that we know what a frame looks like we need a function to initialize the list of frames. This function sets the initial frame name to "initial" and creates a list of frames containing an empty frame. This list is the interpreter frame ring and is not actually circular but is managed as a circular list.

As a final step we update the world from this frame. This has the side-effect of resetting all the important global variables to their initial values.

[emptyInterpreterFrame p552]
[updateFromCurrentInterpreterFrame p554]
[$interpreterFrameName p**??**]
[$interpreterFrameRing p**??**]

— **defun initializeInterpreterFrameRing** —

```
(defun |initializeInterpreterFrameRing| ()
 "Initializing the Interpreter Frame Ring"
 (declare (special |$interpreterFrameName| |$interpreterFrameRing|))
 (setq |$interpreterFrameName| '|initial|)
 (setq |$interpreterFrameRing|
   (list (|emptyInterpreterFrame| |$interpreterFrameName|)))
 (|updateFromCurrentInterpreterFrame|)
 nil)
```

———-

33.5.2 Creating a List of all of the Frame Names

This function simply walks across the frame in the frame ring and returns a list of the name of each frame. [$interpreterFrameRing p??]

— **defun frameNames 0** —

```
(defun |frameNames| ()
 "Creating a List of all of the Frame Names"
 (declare (special |$interpreterFrameRing|))
 (mapcar #'frameName |$interpreterFrameRing|))
```

———-

33.5.3 Get Named Frame Environment (aka Interactive)

If the frame is found we return the environment portion of the frame otherwise we construct an empty environment and return it. The initial values of an empty frame are created here. This function returns a single frame that will be placed in the frame ring. [frameInteractive p??]

— **defun frameEnvironment** —

```
(defun |frameEnvironment| (fname)
 "Get Named Frame Environment (aka Interactive)"
 (let ((frame (|findFrameInRing| fname)))
  (if frame
   (frameInteractive frame)
   (list (list nil)))))
```

———-

33.5.4 Create a new, empty Interpreter Frame

[$HiFiAccess p724]
[$HistList p??]
[$HistListLen p??]
[$HistListAct p??]
[$HistRecord p??]
[$localExposureDataDefault p688]

— **defun emptyInterpreterFrame 0** —

```
(defun |emptyInterpreterFrame| (name)
```

```
 "Create a new, empty Interpreter Frame"
 (declare (special |$HiFiAccess| |$HistList| |$HistListLen| |$HistListAct|
   |$HistRecord| |$localExposureDataDefault|))
 (list name                                  ; frame name
   (list (list nil))                         ; environment
   1                                         ; $IOindex
   |$HiFiAccess|
   |$HistList|
   |$HistListLen|
   |$HistListAct|
   |$HistRecord|
   nil                                       ; $internalHistoryTable
   (copy-seq |$localExposureDataDefault|)))  ; $localExposureData
```

———-

33.5.5 Collecting up the Environment into a Frame

We can collect up all the current environment information into one frame element with this call. It creates a list of the current values of the global variables and returns this as a frame element.

[$interpreterFrameName p**??**]
[$InteractiveFrame p**??**]
[$IOindex p**??**]
[$HiFiAccess p724]
[$HistList p**??**]
[$HistListLen p**??**]
[$HistListAct p**??**]
[$HistRecord p**??**]
[$internalHistoryTable p**??**]
[$localExposureData p688]

— defun createCurrentInterpreterFrame 0 —

```
(defun |createCurrentInterpreterFrame| ()
 "Collecting up the Environment into a Frame"
 (declare (special |$interpreterFrameName| |$InteractiveFrame| |$IOindex|
   |$HiFiAccess| |$HistList| |$HistListLen| |$HistListAct| |$HistRecord|
   |$internalHistoryTable| |$localExposureData|))
 (list
   |$interpreterFrameName|
   |$InteractiveFrame|
   |$IOindex|
   |$HiFiAccess|
   |$HistList|
   |$HistListLen|
```

```
  |$HistListAct|
  |$HistRecord|
  |$internalHistoryTable|
  |$localExposureData|))
```

———-

33.5.6 Update from the Current Frame

The frames are kept on a circular list. The first element on that list is known as "the current frame". This will initialize all of the interesting interpreter data structures from that frame.
[sayMessage p**??**]
[$interpreterFrameRing p**??**]
[$interpreterFrameName p**??**]
[$InteractiveFrame p**??**]
[$IOindex p**??**]
[$HiFiAccess p724]
[$HistList p**??**]
[$HistListLen p**??**]
[$HistListAct p**??**]
[$HistRecord p**??**]
[$internalHistoryTable p**??**]
[$localExposureData p688]
[$frameMessages p732]

— defun updateFromCurrentInterpreterFrame —

```
(defun |updateFromCurrentInterpreterFrame| ()
 "Update from the Current Frame"
 (let (tmp1)
  (declare (special |$interpreterFrameRing| |$interpreterFrameName|
    |$InteractiveFrame| |$IOindex| |$HiFiAccess| |$HistList| |$HistListLen|
    |$HistListAct| |$HistRecord| |$internalHistoryTable| |$localExposureData|
    |$frameMessages|))
  (setq tmp1 (first |$interpreterFrameRing|))
  (setq |$interpreterFrameName| (nth 0 tmp1))
  (setq |$InteractiveFrame|     (nth 1 tmp1))
  (setq |$IOindex|              (nth 2 tmp1))
  (setq |$HiFiAccess|           (nth 3 tmp1))
  (setq |$HistList|             (nth 4 tmp1))
  (setq |$HistListLen|          (nth 5 tmp1))
  (setq |$HistListAct|          (nth 6 tmp1))
  (setq |$HistRecord|           (nth 7 tmp1))
  (setq |$internalHistoryTable| (nth 8 tmp1))
  (setq |$localExposureData|    (nth 9 tmp1))
  (when |$frameMessages|
```

```
  (|sayMessage|
   `("   Current interpreter frame is called"
     ,#(|bright| |$interpreterFrameName|))))))
```

———-

33.5.7 Find a Frame in the Frame Ring by Name

Each frame contains its name as the 0th element. We simply walk all the frames and if we find one we return it. [boot-equal p**??**]
[frameName p548]
[$interpreterFrameRing p**??**]

— **defun findFrameInRing 0** —

```
(defun |findFrameInRing| (name)
 "Find a Frame in the Frame Ring by Name"
 (let (result)
  (declare (special |$interpreterFrameRing|))
  (dolist (frame |$interpreterFrameRing|)
    (when (boot-equal (frameName frame) name)
      (setq result frame)))
  result))
```

———-

33.5.8 Update the Current Interpreter Frame

This function collects the normal contents of the world into a frame object, places it first on the frame list, and then sets the current values of the world from the frame object. [createCurrentInterpreterFrame p553]
[updateFromCurrentInterpreterFrame p554]
[$interpreterFrameRing p**??**]

— **defun updateCurrentInterpreterFrame** —

```
(defun |updateCurrentInterpreterFrame| ()
  "Update the Current Interpreter Frame"
  (declare (special |$interpreterFrameRing|))
  (rplaca |$interpreterFrameRing| (|createCurrentInterpreterFrame|))
  (|updateFromCurrentInterpreterFrame|))
```

———-

33.5.9 Move to the next Interpreter Frame in Ring

This function updates the current frame to make sure all of the current information is recorded. If there are more frame elements in the list then this will destructively move the current frame to the end of the list, that is, assume the frame list reads (1 2 3) this function will destructively change it to (2 3 1). [updateFromCurrentInterpreterFrame p554]
[$interpreterFrameRing p??]

— **defun nextInterpreterFrame** —

```
(defun |nextInterpreterFrame| ()
  "Move to the next Interpreter Frame in Ring"
  (declare (special |$interpreterFrameRing|))
  (when (cdr |$interpreterFrameRing|)
   (setq |$interpreterFrameRing|
    (nconc (cdr |$interpreterFrameRing|) (list (car |$interpreterFrameRing|))))
   (|updateFromCurrentInterpreterFrame|)))
```

————-

33.5.10 Change to the Named Interpreter Frame

[updateCurrentInterpreterFrame p555]
[findFrameInRing p555]
[nremove p??]
[updateFromCurrentInterpreterFrame p554]
[$interpreterFrameRing p??]

— **defun changeToNamedInterpreterFrame** —

```
(defun |changeToNamedInterpreterFrame| (name)
 "Change to the Named Interpreter Frame"
 (let (frame)
 (declare (special |$interpreterFrameRing|))
   (|updateCurrentInterpreterFrame|)
   (setq frame (|findFrameInRing| name))
   (when frame
     (setq |$interpreterFrameRing|
       (cons frame (nremove |$interpreterFrameRing| frame)))
     (|updateFromCurrentInterpreterFrame|))))
```

————-

33.5.11 Move to the previous Interpreter Frame in Ring

[updateCurrentInterpreterFrame p555]
[updateFromCurrentInterpreterFrame p554]
[$interpreterFrameRing p??]

— **defun previousInterpreterFrame** —

```
(defun |previousInterpreterFrame| ()
 "Move to the previous Interpreter Frame in Ring"
 (let (tmp1 l b)
 (declare (special |$interpreterFrameRing|))
  (|updateCurrentInterpreterFrame|)
  (when  (cdr |$interpreterFrameRing|)
    (setq tmp1 (reverse |$interpreterFrameRing|))
    (setq l (car tmp1))
    (setq b (nreverse (cdr tmp1)))
    (setq |$interpreterFrameRing| (nconc (cons l nil) b))
    (|updateFromCurrentInterpreterFrame|))))
```

———-

33.5.12 Add a New Interpreter Frame

[boot-equal p??]
[framename p??]
[throwKeyedMsg p??]
[updateCurrentInterpreterFrame p555]
[initHistList p577]
[emptyInterpreterFrame p552]
[updateFromCurrentInterpreterFrame p554]
[$erase p??]
[histFileName p576]
[$interpreterFrameRing p??]

— **defun addNewInterpreterFrame** —

```
(defun |addNewInterpreterFrame| (name)
 "Add a New Interpreter Frame"
 (declare (special |$interpreterFrameRing|))
 (if (null name)
   (|throwKeyedMsg| 's2iz0018 nil)  ; you must provide a name for new frame
   (progn
     (|updateCurrentInterpreterFrame|)
     (dolist (f |$interpreterFrameRing|)
       (when (boot-equal name (frameName f)) ; existing frame with same name
```

```
        (|throwKeyedMsg| 's2iz0019 (list name))))
    (|initHistList|)
    (setq |$interpreterFrameRing|
      (cons (|emptyInterpreterFrame| name) |$interpreterFrameRing|))
    (|updateFromCurrentInterpreterFrame|)
    ($erase (|histFileName|)))))
```

———-

33.5.13 Close an Interpreter Frame

[framename p**??**]
[throwKeyedMsg p**??**]
[$erase p**??**]
[makeHistFileName p575]
[updateFromCurrentInterpreterFrame p554]
[$interpreterFrameRing p**??**]
[$interpreterFrameName p**??**]

— defun closeInterpreterFrame —

```
(defun |closeInterpreterFrame| (name)
 "Close an Interpreter Frame"
 (declare (special |$interpreterFrameRing| |$interpreterFrameName|))
 (let (ifr found)
    (if (null (cdr |$interpreterFrameRing|))
     (if (and name (not (equal name |$interpreterFrameName|)))
       (|throwKeyedMsg| 's2iz0020 ; 1 frame left. not the correct name.
          (cons |$interpreterFrameName| nil))
       (|throwKeyedMsg| 's2iz0021 nil)) ; only 1 frame left, not closed
     (progn
       (if (null name)
        (setq |$interpreterFrameRing| (cdr |$interpreterFrameRing|))
        (progn
          (setq found nil)
          (setq ifr nil)
          (dolist (f |$interpreterFrameRing|)
            (if (or found (not (equal name (frameName f))))
              (setq ifr (cons f ifr)))
              (setq found t))
          (if (null found)
           (|throwKeyedMsg| 's2iz0022 (cons name nil))
           (progn
             ($erase (|makeHistFileName| name))
             (setq |$interpreterFrameRing| (nreverse ifr))))))
       (|updateFromCurrentInterpreterFrame|)))))
```

———-

33.5.14 Display the Frame Names

[bright p??]
[framename p??]
[sayKeyedMsg p327]
[$interpreterFrameRing p??]

— **defun displayFrameNames** —

```
(defun |displayFrameNames| ()
 "Display the Frame Names"
 (declare (special |$interpreterFrameRing|))
 (let (t1)
  (setq t1
   (mapcar #'(lambda (f) `(|%l| "     " ,@(|bright| (frameName f))))
   |$interpreterFrameRing|))
 (|sayKeyedMsg| 's2iz0024 (list (apply #'append t1)))))
```

———-

33.5.15 Import items from another frame

[member p1042]
[frameNames p552]
[throwKeyedMsg p??]
[boot-equal p??]
[framename p??]
[frameEnvironment p552]
[upcase p??]
[queryUserKeyedMsg p??]
[string2id-n p??]
[importFromFrame p559]
[sayKeyedMsg p327]
[clearCmdParts p501]
[seq p??]
[exit p??]
[putHist p586]
[get p??]
[getalist p??]
[$interpreterFrameRing p??]

— **defun importFromFrame** —

```
(defun |importFromFrame| (args)
 "Import items from another frame"
 (prog (temp1 fname fenv x v props vars plist prop val m)
 (declare (special |$interpreterFrameRing|))
  (when (and args (atom args)) (setq args (cons args nil)))
  (if (null args)
   (|throwKeyedMsg| 'S2IZ0073 nil) ; missing frame name
   (progn
     (setq temp1 args)
     (setq fname (car temp1))
     (setq args (cdr temp1))
     (cond
      ((null (|member| fname (|frameNames|)))
        (|throwKeyedMsg| 'S2IZ0074 (cons fname nil))) ; not frame name
      ((boot-equal fname (frameName (car |$interpreterFrameRing|)))
        (|throwKeyedMsg| 'S2IZ0075 NIL)) ; cannot import from curr frame
      (t
        (setq fenv (|frameEnvironment| fname))
        (cond
         ((null args)
           (setq x
             (upcase (|queryUserKeyedMsg| 'S2IZ0076 (cons fname nil))))
                                        ; import everything?
           (cond
            ((member (string2id-n x 1) '(y yes))
              (setq vars nil)
              (do ((tmp0 (caar fenv) (cdr tmp0)) (tmp1 nil))
                  ((or (atom tmp0)
                       (progn (setq tmp1 (car tmp0)) nil)
                       (progn
                        (progn
                         (setq v (car tmp1))
                         (setq props (cdr tmp1))
                         tmp1)
                        nil))
                   nil)
               (cond
                ((eq v '|--macros|)
                  (do ((tmp2 props (cdr tmp2))
                       (tmp3 nil))
                      ((or (atom tmp2)
                           (progn (setq tmp3 (car tmp2)) nil)
                           (progn
                            (progn (setq m (car tmp3)) tmp3)
                            nil))
                       nil)
                    (setq vars (cons m vars))))
                (t (setq vars (cons v vars)))))
             (|importFromFrame| (cons fname vars)))
            (t
```

```
        (|sayKeyedMsg| 'S2IZ0077 (cons fname nil)))))
      (t
       (do ((tmp4 args (cdr tmp4)) (v nil))
           ((or (atom tmp4) (progn (setq v (car tmp4)) nil)) nil)
        (seq
         (exit
          (progn
           (setq plist (getalist (caar fenv) v))
           (cond
            (plist
             (|clearCmdParts| (cons '|propert| (cons v nil)))
             (do ((tmp5 plist (cdr tmp5)) (tmp6 nil))
                 ((or (atom tmp5)
                      (progn (setq tmp6 (car tmp5)) nil)
                      (progn
                       (progn
                        (setq prop (car tmp6))
                        (setq val (cdr tmp6))
                        tmp6)
                       nil))
                  nil)
              (seq
               (exit (|putHist| v prop val |$InteractiveFrame|)))))
            ((setq m (|get| '|--macros--| v fenv))
             (|putHist| '|--macros--| v m |$InteractiveFrame|))
            (t
             (|sayKeyedMsg| 'S2IZ0079 ; frame not found
              (cons v (cons fname nil)))))))))
        (|sayKeyedMsg| 'S2IZ0078 ; import complete
         (cons fname nil))))))))))
```

———-

33.5.16 The top level frame command

[frameSpad2Cmd p562]

— **defun frame** —

```
(defun |frame| (l)
 "The top level frame command"
 (|frameSpad2Cmd| l))
```

———-

33.5.17 The top level frame command handler

[throwKeyedMsg p??]
[helpSpad2Cmd p568]
[selectOptionLC p475]
[qcdr p??]
[qcar p??]
[object2Identifier p??]
[frameSpad2Cmd drop (vol9)]
[closeInterpreterFrame p558]
[import p??]
[importFromFrame p559]
[last p??]
[previousInterpreterFrame p557]
[names p??]
[displayFrameNames p559]
[new p??]
[addNewInterpreterFrame p557]
[next p38]
[nextInterpreterFrame p556]
[$options p??]

— defun frameSpad2Cmd —

```
(defun |frameSpad2Cmd| (args)
 "The top level frame command handler"
 (let (frameArgs arg a)
  (declare (special |$options|))
  (setq frameArgs '(|drop| |import| |last| |names| |new| |next|))
  (cond
   (|$options|
    (|throwKeyedMsg| 'S2IZ0016 ; frame command does not take options
     (cons ")frame" nil)))
   ((null args) (|helpSpad2Cmd| (cons '|frame| nil)))
   (t
     (setq arg (|selectOptionLC| (car args) frameArgs '|optionError|))
     (setq args (cdr args))
     (when (and (consp args)
            (eq (qcdr args) nil)
            (progn (setq a (qcar args)) t))
        (setq args a))
     (when (atom args) (setq args (|object2Identifier| args)))
     (case arg
      (|drop|
        (if (and args (consp args))
          (|throwKeyedMsg| 'S2IZ0017 ; not a valid frame name
             (cons args nil))
          (|closeInterpreterFrame| args)))
```

```
(|import| (|importFromFrame| args))
(|last|   (|previousInterpreterFrame|))
(|names|  (|displayFrameNames|))
(|new|
  (if (and args (consp args))
    (|throwKeyedMsg| 'S2IZ0017 ; not a valid frame name
       (cons args nil))
    (|addNewInterpreterFrame| args)))
(|next| (|nextInterpreterFrame|))
(t nil))))))
```

———-

33.6 Frame File Messages

— Frame File Messages —

```
S2IZ0016
 The %1b system command takes arguments but no options.
S2IZ0017
 %1b is not a valid frame name
S2IZ0018
 You must provide a name for the new frame.
S2IZ0019
 You cannot use the name %1b for a new frame because an existing
 frame already has that name.
S2IZ0020
 There is only one frame active and therefore that cannot be closed.
 Furthermore, the frame name you gave is not the name of the current frame.
 The current frame is called %1b .
S2IZ0021
 The current frame is the only active one.  Issue %b )clear all %d to
 clear its contents.
S2IZ0022
 There is no frame called %1b and so your command cannot be
 processed.
S2IZ0024
 The names of the existing frames are: %1 %l
 The current frame is the first one listed.
S2IZ0073
 %b )frame import %d must be followed by the frame name. The names
 of objects in that frame can then optionally follow the frame name.
 For example,
 %ceon %b )frame import calculus %d %ceoff
 imports all objects in the %b calculus %d frame, and
 %ceon %b )frame import calculus epsilon delta %d %ceoff
```

```
 imports the objects named %b epsilon %d and %b delta %d from the
 frame %b calculus %d .
 Please note that if the current frame contained any information
 about objects with these names, then that information would be
 cleared before the import took place.
S2IZ0074
 You cannot import anything from the frame %1b because that is not
 the name of an existing frame.
S2IZ0075
 You cannot import from the current frame (nor is there a need!).
S2IZ0076
 User verification required:
 do you really want to import everything from the frame %1b ?
 If so, please enter %b y %d or %b yes %d :
S2IZ0077
 On your request, AXIOM will not import everything from frame %1b.
S2IZ0078
 Import from frame %1b is complete. Please issue %b )display all %d
 if you wish to see the contents of the current frame.
S2IZ0079
 AXIOM cannot import %1b from frame %2b because it cannot be found.
```

———-

Chapter 34

)help help page Command

34.1 help help page man page

— help.help —

```
====================================================================
A.12.  )help
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )help
  - )help commandName
  - )help syntax

Command Description:

This command displays help information about system commands. If you issue

)help

then this very text will be shown. You can also give the name or abbreviation
of a system command to display information about it. For example,

)help clear

will display the description of the )clear system command.

The command
```

```
)help syntax

will give further information about the Axiom language syntax.

All this material is available in the AXIOM User Guide and in HyperDoc. In
HyperDoc, choose the Commands item from the Reference menu.

========================================================================
A.1.  Introduction
========================================================================


System commands are used to perform AXIOM environment management. Among the
commands are those that display what has been defined or computed, set up
multiple logical AXIOM environments (frames), clear definitions, read files
of expressions and commands, show what functions are available, and terminate
AXIOM.

Some commands are restricted: the commands

)set userlevel interpreter
)set userlevel compiler
)set userlevel development

set the user-access level to the three possible choices. All commands are
available at development level and the fewest are available at interpreter
level. The default user-level is interpreter. In addition to the )set command
(discussed in description of command )set ) you can use the HyperDoc settings
facility to change the user-level. Click on [Settings] here to immediately go
to the settings facility.

Each command listing begins with one or more syntax pattern descriptions plus
examples of related commands. The syntax descriptions are intended to be easy
to read and do not necessarily represent the most compact way of specifying
all possible arguments and options; the descriptions may occasionally be
redundant.

All system commands begin with a right parenthesis which should be in the
first available column of the input line (that is, immediately after the
input prompt, if any). System commands may be issued directly to AXIOM or be
included in .input files.

A system command argument is a word that directly follows the command name
and is not followed or preceded by a right parenthesis. A system command
option follows the system command and is directly preceded by a right
parenthesis. Options may have arguments: they directly follow the option.
This example may make it easier to remember what is an option and what is an
argument:

         )syscmd arg1 arg2 )opt1 opt1arg1 opt1arg2 )opt2 opt2arg1 ...
```

```
In the system command descriptions, optional arguments and options are
enclosed in brackets (``['' and ``]''). If an argument or option name is in
italics, it is meant to be a variable and must have some actual value
substituted for it when the system command call is made. For example, the
syntax pattern description

)read fileName []quietly]

would imply that you must provide an actual file name for fileName but need
not use the )quietly option. Thus

)read matrix.input

is a valid instance of the above pattern.

System command names and options may be abbreviated and may be in upper or
lower case. The case of actual arguments may be significant, depending on the
particular situation (such as in file names). System command names and
options may be abbreviated to the minimum number of starting letters so that
the name or option is unique. Thus

)s Integer

is not a valid abbreviation for the )set command, because both )set and )show
begin with the letter ``s''. Typically, two or three letters are sufficient
for disambiguating names. In our descriptions of the commands, we have used
no abbreviations for either command names or options.

In some syntax descriptions we use a vertical line ``|'' to indicate that you
must specify one of the listed choices. For example, in

)set output fortran on | off

only on and off are acceptable words for following boot. We also sometimes
use ``...'' to indicate that additional arguments or options of the listed
form are allowed. Finally, in the syntax descriptions we may also list the
syntax of related commands.

========================================================================
Other help topics
========================================================================
Available help topics are:

abbreviations assignment blocks     browse     boot       cd
clear         clef       close      collection compile    describe
display       edit       fin        for        frame      help
history       if         iterate    leave      library    lisp
load          ltrace     parallel   pquit      quit       read
repeat        savesystem set        show       spool      suchthat
```

```
synonym       system     syntax     trace      undo       what
while

Available algebra help topics are:
```

———-

34.2 Functions

34.2.1 The top level help command

[helpSpad2Cmd p568]

— **defun help** —

```
(defun |help| (l)
 "The top level help command"
 (|helpSpad2Cmd| l))
```

———-

34.2.2 The top level help command handler

[newHelpSpad2Cmd p568]
[sayKeyedMsg p327]

— **defun helpSpad2Cmd** —

```
(defun |helpSpad2Cmd| (args)
 "The top level help command handler"
 (unless (|newHelpSpad2Cmd| args)
  (|sayKeyedMsg| 's2iz0025 (cons args nil))))
```

———-

34.2.3 defun newHelpSpad2Cmd

[makeInputFilename p973]
[obey p**??**]
[concat p1041]
[namestring p1034]

[make-instream p971]
[say p**??**]
[poundsign p**??**]
[sayKeyedMsg p327]
[pname p1039]
[selectOptionLC p475]
[$syscommands p440]
[$useFullScreenHelp p723]

— **defun newHelpSpad2Cmd** —

```
(defun |newHelpSpad2Cmd| (args)
 (let (sarg arg narg helpfile filestream line)
 (declare (special $syscommands |$useFullScreenHelp|))
  (when (null args) (setq args (list '?)))
  (if (> (|#| args) 1)
   (|sayKeyedMsg| 's2iz0026 nil)
   (progn
    (setq sarg (pname (car args)))
    (cond
     ((string= sarg "?") (setq args (list '|help|)))
     ((string= sarg "%") (setq args (list '|history|)))
     ((string= sarg "%%") (setq args (list '|history|)))
     (t nil))
    (setq arg (|selectOptionLC| (car args) $syscommands nil))
    (cond ((null arg) (setq arg (car args))))
    (setq narg (pname arg))
    (cond
     ((null (setq helpfile (makeInputFilename (list narg "help"))))
       nil)
     (|$useFullScreenHelp|
      (obey (concat "$AXIOM/lib/SPADEDIT " (|namestring| helpfile))) t)
     (t
      (setq filestream (make-instream helpfile))
      (do ((line (|read-line| filestream nil) (|read-line| filestream nil)))
          ((null line) (shut filestream))
       (say line))))))))
```

———-

Chapter 35

)history help page Command

35.1 history help page man page

— history.help —

```
====================================================================
A.13.  )history
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )history )on
  - )history )off
  - )history )write historyInputFileName
  - )history )show [n] [both]
  - )history )save savedHistoryName
  - )history )restore [savedHistoryName]
  - )history )reset
  - )history )change n
  - )history )memory
  - )history )file
  - %
  - %%(n)
  - )set history on | off

Command Description:

The history facility within AXIOM allows you to restore your environment to
that of another session and recall previous computational results. Additional
commands allow you to review previous input lines and to create an .input
```

file of the lines typed to AXIOM.

AXIOM saves your input and output if the history facility is turned on (which is the default). This information is saved if either of

```
)set history on
)history )on
```

has been issued. Issuing either

```
)set history off
)history )off
```

will discontinue the recording of information.

Whether the facility is disabled or not, the value of % in AXIOM always refers to the result of the last computation. If you have not yet entered anything, % evaluates to an object of type Variable('%). The function %% may be used to refer to other previous results if the history facility is enabled. In that case, %%(n) is the output from step n if n > 0. If n < 0, the step is computed relative to the current step. Thus %%(-1) is also the previous step, %%(-2), is the step before that, and so on. If an invalid step number is given, AXIOM will signal an error.

The environment information can either be saved in a file or entirely in memory (the default). Each frame (description of command)frame) has its own history database. When it is kept in a file, some of it may also be kept in memory for efficiency. When the information is saved in a file, the name of the file is of the form FRAME.axh where ``FRAME'' is the name of the current frame. The history file is placed in the current working directory (see description of command)cd). Note that these history database files are not text files (in fact, they are directories themselves), and so are not in human-readable format.

The options to the)history command are as follows:

)change n
  will set the number of steps that are saved in memory to n. This option only has effect when the history data is maintained in a file. If you have issued)history)memory (or not changed the default) there is no need to use)history)change.

)on
  will start the recording of information. If the workspace is not empty, you will be asked to confirm this request. If you do so, the workspace will be cleared and history data will begin being saved. You can also turn the facility on by issuing)set history on.

)off
  will stop the recording of information. The)history)show command will

```
  not work after issuing this command. Note that this command may be issued
  to save time, as there is some performance penalty paid for saving the
  environment data. You can also turn the facility off by issuing )set
  history off.

)file
  indicates that history data should be saved in an external file on disk.

)memory
  indicates that all history data should be kept in memory rather than
  saved in a file. Note that if you are computing with very large objects
  it may not be practical to kept this data in memory.

)reset
  will flush the internal list of the most recent workspace calculations so
  that the data structures may be garbage collected by the underlying Lisp
  system. Like )history )change, this option only has real effect when
  history data is being saved in a file.

)restore [savedHistoryName]
  completely clears the environment and restores it to a saved session, if
  possible. The )save option below allows you to save a session to a file
  with a given name. If you had issued )history )save jacobi the command
  )history )restore jacobi would clear the current workspace and load the
  contents of the named saved session. If no saved session name is
  specified, the system looks for a file called last.axh.

)save savedHistoryName
  is used to save a snapshot of the environment in a file. This file is
  placed in the current working directory (see description of command )cd
  ). Use )history )restore to restore the environment to the state
  preserved in the file. This option also creates an input file containing
  all the lines of input since you created the workspace frame (for
  example, by starting your AXIOM session) or last did a )clear all or
  )clear completely.

)show [n] [both]
  can show previous input lines and output results. )show will display up
  to twenty of the last input lines (fewer if you haven't typed in twenty
  lines). )show n will display up to n of the last input lines. )show both
  will display up to five of the last input lines and output results. )show
  n both will display up to n of the last input lines and output results.

)write historyInputFile
  creates an .input file with the input lines typed since the start of the
  session/frame or the last )clear all or )clear completely. If
  historyInputFileName does not contain a period (``.'') in the filename,
  .input is appended to it. For example, )history )write chaos and )history
  )write chaos.input both write the input lines to a file called
  chaos.input in your current working directory. If you issued one or more
```

```
    )undo commands, )history )write eliminates all input lines backtracked
    over as a result of )undo. You can edit this file and then use )read to
    have AXIOM process the contents.

Also See:
o )frame
o )read
o )set
o )undo
```

———-

[1]

History recording is done in two different ways:

- all changes in variable bindings (i.e. previous values) are written to `$HistList`, which is a circular list
- all new bindings (including the binding to `%`) are written to a file called histFileName() one older session is accessible via the file `$oldHistFileName()`

35.2 Initialized history variables

The following global variables are used:

`$HistList`, `$HistListLen` and `$HistListAct` which is the actual number of "undoable" steps)

`$HistRecord` collects the input line, all variable bindings and the output of a step, before it is written to the file histFileName().

`$HiFiAccess` is a flag, which is reset by)history)off

The result of step n can be accessed by `%n`, which is translated into a call of fetchOutput(n). The updateHist is called after every interpreter step. The putHist function records all changes in the environment to `$HistList` and `$HistRecord`.

35.2.1 defvar $oldHistoryFileName

— **initvars** —

```
(defvar |$oldHistoryFileName| '|last| "vm/370 filename name component")
```

———-

[1] "frame" (33.5.16 p 561) "read" (43.1.1 p 638) "set" (45.36.1 p 799) "undo" (52.4.6 p 912)

35.2.2 defvar $historyFileType

— **initvars** —

```
(defvar |$historyFileType| '|axh|     "vm/370 filename type component")
```

————-

35.2.3 defvar $historyDirectory

— **initvars** —

```
(defvar |$historyDirectory| 'A        "vm/370 filename disk component")
```

————-

35.2.4 defvar $useInternalHistoryTable

— **initvars** —

```
(defvar |$useInternalHistoryTable| t  "t means keep history in core")
```

————-

35.3 Data Structures

35.4 Functions

35.4.1 defun makeHistFileName

[makePathname p1036]

— **defun makeHistFileName** —

```
(defun |makeHistFileName| (fname)
 (|makePathname| fname |$historyFileType| |$historyDirectory|))
```

————-

35.4.2 defun oldHistFileName

[makeHistFileName p575]
[$oldHistoryFileName p574]

— **defun oldHistFileName** —

```
(defun |oldHistFileName| ()
 (declare (special |$oldHistoryFileName|))
 (|makeHistFileName| |$oldHistoryFileName|))
```

————-

35.4.3 defun histFileName

[makeHistFileName p575]
[$interpreterFrameName p??]

— **defun histFileName** —

```
(defun |histFileName| ()
 (declare (special |$interpreterFrameName|))
 (|makeHistFileName| |$interpreterFrameName|))
```

————-

35.4.4 defun histInputFileName

[makePathname p1036]
[$interpreterFrameName p??]
[$historyDirectory p575]

— **defun histInputFileName** —

```
(defun |histInputFileName| (fn)
 (declare (special |$interpreterFrameName| |$historyDirectory|))
 (if (null fn)
   (|makePathname| |$interpreterFrameName| 'input |$historyDirectory|)
   (|makePathname| fn 'input |$historyDirectory|)))
```

————-

35.4.5 defun initHist

[initHistList p577]
[oldHistFileName p576]
[histFileName p576]
[histFileErase p613]
[makeInputFilename p973]
[$replace p??]
[$useInternalHistoryTable p575]
[$HiFiAccess p724]

— **defun initHist** —

```
(defun |initHist| ()
 (let (oldFile newFile)
 (declare (special |$useInternalHistoryTable| |$HiFiAccess|))
   (if |$useInternalHistoryTable|
     (|initHistList|)
     (progn
       (setq oldFile (|oldHistFileName|))
       (setq newFile (|histFileName|))
       (|histFileErase| oldFile)
       (when (makeInputFilename newFile) (replaceFile oldFile newFile))
       (setq |$HiFiAccess| t)
       (|initHistList|)))))
```

———-

35.4.6 defun initHistList

[$HistListLen p??]
[$HistList p??]
[$HistListAct p??]
[$HistRecord p??]

— **defun initHistList** —

```
(defun |initHistList| ()
 (let (li)
 (declare (special |$HistListLen| |$HistList| |$HistListAct| |$HistRecord|))
   (setq |$HistListLen| 20)
   (setq |$HistList| (list nil))
   (setq li |$HistList|)
   (do ((i 1 (1+ i)))
       ((> i |$HistListLen|) nil)
     (setq li (cons nil li)))
```

```
   (rplacd |$HistList| li)
   (setq |$HistListAct| 0)
   (setq |$HistRecord| nil)))
```

———-

35.4.7 The top level history command

[sayKeyedMsg p327]
[historySpad2Cmd p578]
[$options p??]

— **defun history** —

```
(defun |history| (l)
 "The top level history command"
 (declare (special |$options|))
 (if (or l (null |$options|))
  (|sayKeyedMsg| 's2ih0006 nil) ; syntax error
  (|historySpad2Cmd|)))
```

———-

35.4.8 The top level history command handler

[selectOptionLC p475]
[member p1042]
[sayKeyedMsg p327]
[initHistList p577]
[upcase p??]
[queryUserKeyedMsg p??]
[string2id-n p??]
[histFileErase p613]
[histFileName p576]
[clearSpad2Cmd p496]
[disableHist p599]
[setHistoryCore p580]
[resetInCoreHist p584]
[saveHistory p591]
[showHistory p??]
[changeHistListLen p585]
[restoreHistory p593]
[writeInputLines p583]
[seq p??]

[exit p**??**]
[$options p**??**]
[$HiFiAccess p724]
[$IOindex p**??**]

— defun historySpad2Cmd —

```
(defun |historySpad2Cmd| ()
 "The top level history command handler"
 (let (histOptions opts opt optargs x)
 (declare (special |$options| |$HiFiAccess| |$IOindex|))
  (setq histOptions
    '(|on| |off| |yes| |no| |change| |reset| |restore| |write|
      |save| |show| |file| |memory|))
  (setq opts
   (prog (tmp1)
     (setq tmp1 nil)
     (return
      (do ((tmp2 |$options| (cdr tmp2)) (tmp3 nil))
          ((or (atom tmp2)
               (progn
                 (setq tmp3 (car tmp2))
                 nil)
               (progn
                (progn
                 (setq opt (car tmp3))
                 (setq optargs (cdr tmp3))
                 tmp3)
                nil))
             (nreverse0 tmp1))
        (setq tmp1
         (cons
           (cons
             (|selectOptionLC| opt histOptions '|optionError|)
             optargs)
           tmp1))))))
  (do ((tmp4 opts (cdr tmp4)) (tmp5 nil))
      ((or (atom tmp4)
           (progn
             (setq tmp5 (car tmp4))
             nil)
           (progn
            (progn
             (setq opt (car tmp5))
             (setq optargs (cdr tmp5))
             tmp5)
            nil))
       nil)
    (seq
```

```
    (exit
     (cond
      ((|member| opt '(|on| |yes|))
         (cond
          (|$HiFiAccess|
            (|sayKeyedMsg| 'S2IH0007 nil)) ; history already on
          ((eql |$IOindex| 1)
            (setq |$HiFiAccess| t)
            (|initHistList|)
            (|sayKeyedMsg| 'S2IH0008 nil)) ; history now on
          (t
            (setq x ; really want to turn history on?
              (upcase (|queryUserKeyedMsg| 'S2IH0009 nil)))
            (cond
             ((member (string2id-n x 1) '(Y YES))
               (|histFileErase| (|histFileName|))
               (setq |$HiFiAccess| t)
               (setq |$options| nil)
               (|clearSpad2Cmd| '(|all|))
               (|sayKeyedMsg| 'S2IH0008 nil) ; history now on
               (|initHistList|))
             (t
               (|sayKeyedMsg| 'S2IH0010 nil))))))) ; history still off
      ((|member| opt '(|off| |no|))
        (cond
         ((null |$HiFiAccess|)
           (|sayKeyedMsg| 'S2IH0011 nil)) ; history already off
         (t
           (setq |$HiFiAccess| nil)
           (|disableHist|)
           (|sayKeyedMsg| 'S2IH0012 nil)))) ; history now off
      ((eq opt '|file|)    (|setHistoryCore| nil))
      ((eq opt '|memory|)  (|setHistoryCore| t))
      ((eq opt '|reset|)   (|resetInCoreHist|))
      ((eq opt '|save|)    (|saveHistory| optargs))
      ((eq opt '|show|)    (|showHistory| optargs))
      ((eq opt '|change|)  (|changeHistListLen| (car optargs)))
      ((eq opt '|restore|) (|restoreHistory| optargs))
      ((eq opt '|write|)   (|writeInputLines| optargs 1))))))
 '|done|))
```

———-

35.4.9 defun setHistoryCore

We case on the inCore argument value

If history is already on and is kept in the same location as requested (file or memory)

then complain.

If history is not in use then start using the file or memory as requested. This is done by simply setting the `$useInternalHistoryTable` to the requested value, where T means use memory and NIL means use a file. We tell the user.

If history should be in memory, that is inCore is not NIL, and the history file already contains information we read the information from the file, store it in memory, and erase the history file. We modify `$useInternalHistoryTable` to T to indicate that we're maintining the history in memory and tell the user.

Otherwise history must be on and in memory. We erase any old history file and then write the in-memory history to a new file

[boot-equal p**??**]
[sayKeyedMsg p327]
[rkeyids p**??**]
[histFileName p576]
[readHiFi p597]
[disableHist p599]
[histFileErase p613]
[rdefiostream p**??**]
[spadrwrite p601]
[object2Identifier p**??**]
[rshut p**??**]
[$useInternalHistoryTable p575]
[$internalHistoryTable p**??**]
[$HiFiAccess p724]
[$IOindex p**??**]

— defun setHistoryCore —

```
(defun |setHistoryCore| (inCore)
 (let (l vec str n rec)
 (declare (special |$useInternalHistoryTable| |$internalHistoryTable|
     |$HiFiAccess| |$IOindex|))
  (cond
   ((boot-equal inCore |$useInternalHistoryTable|)
     (if inCore
        (|sayKeyedMsg| 's2ih0030 nil) ; memory history already in use
        (|sayKeyedMsg| 's2ih0029 nil))) ; file history already in use
   ((null |$HiFiAccess|)
     (setq |$useInternalHistoryTable| inCore)
     (if inCore
        (|sayKeyedMsg| 's2ih0032 nil) ; use memory history
        (|sayKeyedMsg| 's2ih0031 nil))) ; use file history
   (inCore
     (setq |$internalHistoryTable| nil)
     (cond
```

```
     ((not (eql |$IOindex| 0))
       (setq l (length (rkeyids (|histFileName|))))
       (do ((i 1 (1+ i)))
           ((> i l) nil)
         (setq vec (unwind-protect (|readHiFi| i) (|disableHist|)))
         (setq |$internalHistoryTable|
           (cons (cons i vec) |$internalHistoryTable|)))
       (|histFileErase| (|histFileName|))))
    (setq |$useInternalHistoryTable| t)
    (|sayKeyedMsg| 'S2IH0032 nil)) ; use memory history
   (t
    (setq |$HiFiAccess| nil)
    (|histFileErase| (|histFileName|))
    (setq str
     (rdefiostream
      (cons
        '(mode . output)
        (cons
          (cons 'file (|histFileName|))
          nil))))
    (do ((tmp0 (reverse |$internalHistoryTable|) (cdr tmp0))
         (tmp1 nil))
        ((or (atom tmp0)
             (progn
               (setq tmp1 (car tmp0))
               nil)
             (progn
              (progn
               (setq n (car tmp1))
               (setq rec (cdr tmp1))
               tmp1)
              nil))
             nil)
      (spadrwrite (|object2Identifier| n) rec str))
    (rshut str)
    (setq |$HiFiAccess| t)
    (setq |$internalHistoryTable| nil)
    (setq |$useInternalHistoryTable| nil)
    (|sayKeyedMsg| 's2ih0031 nil))))) ; use file history
```

———-

35.4.10 defvar $underbar

Also used in the output routines.

— **initvars** —

```
(defvar underbar "_")
```

———-

35.4.11 defun writeInputLines

[sayKeyedMsg p327]
[throwKeyedMsg p??]
[size p1039]
[spaddifference p??]
[concat p1041]
[substring p??]
[readHiFi p597]
[histInputFileName p576]
[histFileErase p613]
[defiostream p972]
[namestring p1034]
[shut p972]
[underbar p582]
[$HiFiAccess p724]
[$IOindex p??]

— **defun writeInputLines** —

```
(defun |writeInputLines| (fn initial)
 (let (maxn breakChars vecl k svec done n lineList file inp)
 (declare (special underbar |$HiFiAccess| |$IOindex|))
  (cond
   ((null |$HiFiAccess|) (|sayKeyedMsg| 's2ih0013 nil)) ; history is not on
   ((null fn) (|throwKeyedMsg| 's2ih0038 nil)) ; missing file name
   (t
     (setq maxn 72)
     (setq breakChars (cons '| | (cons '+ nil)))
     (do ((tmp0 (spaddifference |$IOindex| 1))
          (i initial (+ i 1)))
         ((> i tmp0) nil)
       (setq vecl (car (|readHiFi| i)))
       (when (stringp vecl) (setq vecl (cons vecl nil)))
       (dolist (vec vecl)
         (setq n (size vec))
         (do ()
             ((null (> n maxn)) nil)
             (setq done nil)
             (do ((j 1 (1+ j)))
                 ((or (> j maxn) (null (null done))) nil)
               (setq k (spaddifference (1+ maxn) j))
               (when (member (elt vec k) breakChars)
```

```
                (setq svec (concat (substring vec 0 (1+ k)) underbar))
                (setq lineList (cons svec lineList))
                (setq done t)
                (setq vec (substring vec (1+ k) nil))
                (setq n (size vec))))
            (when done (setq n 0)))
        (setq lineList (cons vec lineList))))
    (setq file (|histInputFileName| fn))
    (|histFileErase| file)
    (setq inp
     (defiostream
      (cons
       '(mode . output)
       (cons (cons 'file file) nil)) 255 0))
    (dolist (x (|removeUndoLines| (nreverse lineList)))
      (write-line x inp))
    (cond
     ((not (eq fn '|redo|))
       (|sayKeyedMsg| 's2ih0014 ; edit this file to see input lines
         (list (|namestring| file)))))
    (shut inp)
    nil))))
```

———-

35.4.12 defun resetInCoreHist

[$HistListAct p??]
[$HistListLen p??]
[$HistList p??]

— **defun resetInCoreHist** —

```
(defun |resetInCoreHist| ()
 (declare (special |$HistListAct| |$HistListLen| |$HistList|))
 (setq |$HistListAct| 0)
 (do ((i 1 (1+ i)))
     ((> i |$HistListLen|) nil)
   (setq |$HistList| (cdr |$HistList|))
   (rplaca |$HistList| nil)))
```

———-

35.4.13 defun changeHistListLen

[sayKeyedMsg p327]
[spaddifference p??]
[$HistListLen p??]
[$HistList p??]
[$HistListAct p??]

— **defun changeHistListLen** —

```
(defun |changeHistListLen| (n)
 (let (dif l)
 (declare (special |$HistListLen| |$HistList| |$HistListAct|))
 (if (null (integerp n))
   (|sayKeyedMsg| 's2ih0015 (list n)) ; only positive integers
   (progn
     (setq dif (spaddifference n |$HistListLen|))
     (setq |$HistListLen| n)
     (setq l (cdr |$HistList|))
     (cond
      ((> dif 0)
        (do ((i 1 (1+ i)))
            ((> i dif) nil)
           (setq l (cons nil l))))
      ((minusp dif)
        (do ((tmp0 (spaddifference dif))
             (i 1 (1+ i)))
            ((> i tmp0) nil)
           (setq l (cdr l)))
        (cond
         ((> |$HistListAct| n) (setq |$HistListAct| n))
         (t nil))))
     (rplacd |$HistList| l)
     '|done|))))
```

———-

35.4.14 defun updateHist

[startTimingProcess p??]
[updateInCoreHist p586]
[writeHiFi p598]
[disableHist p599]
[updateCurrentInterpreterFrame p555]
[stopTimingProcess p??]
[$IOindex p??]
[$HiFiAccess p724]

[$HistRecord p??]
[$mkTestInputStack p??]
[$currentLine p??]

— **defun updateHist** —

```
(defun |updateHist| ()
 (declare (special |$IOindex| |$HiFiAccess| |$HistRecord| |$mkTestInputStack|
   |$currentLine|))
 (when |$IOindex|
   (|startTimingProcess| '|history|)
   (|updateInCoreHist|)
   (when |$HiFiAccess|
      (unwind-protect (|writeHiFi|) (|disableHist|))
      (setq |$HistRecord| nil))
   (incf |$IOindex|)
   (|updateCurrentInterpreterFrame|)
   (setq |$mkTestInputStack| nil)
   (setq |$currentLine| nil)
   (|stopTimingProcess| '|history|)))
```

————-

35.4.15 defun updateInCoreHist

[$HistList p??]
[$HistListLen p??]
[$HistListAct p??]

— **defun updateInCoreHist** —

```
(defun |updateInCoreHist| ()
 (declare (special |$HistList| |$HistListLen| |$HistListAct|))
  (setq |$HistList| (cdr |$HistList|))
  (rplaca |$HistList| nil)
  (when (> |$HistListLen| |$HistListAct|)
    (setq |$HistListAct| (1+ |$HistListAct|))))
```

————-

35.4.16 defun putHist

[recordOldValue p588]
[get p??]
[recordNewValue p587]

[putIntSymTab p**??**]
[$HiFiAccess p724]

— **defun putHist** —

```
(defun |putHist| (x prop val e)
  (declare (special |$HiFiAccess|))
  (when (null (eq x '%)) (|recordOldValue| x prop (|get| x prop e)))
  (when |$HiFiAccess| (|recordNewValue| x prop val))
  (|putIntSymTab| x prop val e))
```

———-

35.4.17 defun recordNewValue

[startTimingProcess p**??**]
[recordNewValue0 p587]
[stopTimingProcess p**??**]

— **defun recordNewValue** —

```
(defun |recordNewValue| (x prop val)
  (|startTimingProcess| '|history|)
  (|recordNewValue0| x prop val)
  (|stopTimingProcess| '|history|))
```

———-

35.4.18 defun recordNewValue0

[assq p1044]
[$HistRecord p**??**]

— **defun recordNewValue0** —

```
(defun |recordNewValue0| (x prop val)
 (let (p1 p2 p)
 (declare (special |$HistRecord|))
   (if (setq p1 (assq x |$HistRecord|))
    (if (setq p2 (assq prop (cdr p1)))
      (rplacd p2 val)
      (rplacd p1 (cons (cons prop val) (cdr p1))))
    (progn
      (setq p (cons x (list (cons prop val))))
```

```
      (setq |$HistRecord| (cons p |$HistRecord|))))))
```

———-

35.4.19 defun recordOldValue

[startTimingProcess p**??**]
[recordOldValue0 p588]
[stopTimingProcess p**??**]
[assq p1044]

— **defun recordOldValue** —

```
(defun |recordOldValue| (x prop val)
  (|startTimingProcess| '|history|)
  (|recordOldValue0| x prop val)
  (|stopTimingProcess| '|history|))
```

———-

35.4.20 defun recordOldValue0

[$HistList p**??**]

— **defun recordOldValue0** —

```
(defun |recordOldValue0| (x prop val)
 (let (p1 p)
 (declare (special |$HistList|))
 (when (setq p1 (assq x (car |$HistList|)))
   (when (null (assq prop (cdr p1)))
      (rplacd p1 (cons (cons prop val) (cdr p1)))))
 (setq p (cons x (list (cons prop val))))
 (rplaca |$HistList| (cons p (car |$HistList|)))))
```

———-

35.4.21 defun undoInCore

[undoChanges p589]
[spaddifference p**??**]
[readHiFi p597]

[disableHist p599]
[assq p1044]
[sayKeyedMsg p327]
[putHist p586]
[updateHist p585]
[$HistList p??]
[$HistListLen p??]
[$IOindex p??]
[$HiFiAccess p724]
[$InteractiveFrame p??]

— **defun undoInCore** —

```
(defun |undoInCore| (n)
 (let (li vec p p1 val)
 (declare (special |$HistList| |$HistListLen| |$IOindex| |$HiFiAccess|
    |$InteractiveFrame|))
  (setq li |$HistList|)
  (do ((i n (+ i 1)))
      ((> i |$HistListLen|) nil)
   (setq li (cdr li)))
  (|undoChanges| li)
  (setq n (spaddifference (spaddifference |$IOindex| n) 1))
  (and
    (> n 0)
    (if |$HiFiAccess|
     (progn
       (setq vec (cdr (unwind-protect (|readHiFi| n) (|disableHist|))))
       (setq val
        (and
         (setq p (assq '% vec))
         (setq p1 (assq '|value| (cdr p)))
         (cdr p1))))
     (|sayKeyedMsg| 's2ih0019 (cons n nil)))) ; no history file
  (setq |$InteractiveFrame| (|putHist| '% '|value| val |$InteractiveFrame|))
  (|updateHist|)))
```

———-

35.4.22 defun undoChanges

[boot-equal p??]
[undoChanges p589]
[putHist p586]
[$HistList p??]
[$InteractiveFrame p??]

— defun undoChanges —

```
(defun |undoChanges| (li)
 (let (x)
 (declare (special |$HistList| |$InteractiveFrame|))
  (when (null (boot-equal (cdr li) |$HistList|)) (|undoChanges| (cdr li)))
  (dolist (p1 (car li))
    (setq x (car p1))
    (dolist (p2 (cdr p1))
     (|putHist| x (car p2) (cdr p2) |$InteractiveFrame|)))))
```

———-

35.4.23 defun undoFromFile

[seq p**??**]
[exit p**??**]
[recordOldValue p588]
[recordNewValue p587]
[readHiFi p597]
[disableHist p599]
[putHist p586]
[assq p1044]
[updateHist p585]
[$InteractiveFrame p**??**]
[$HiFiAccess p724]

— defun undoFromFile —

```
(defun |undoFromFile| (n)
 (let (varl prop vec x p p1 val)
 (declare (special |$InteractiveFrame| |$HiFiAccess|))
  (do ((tmp0 (caar |$InteractiveFrame|) (cdr tmp0)) (tmp1 nil))
      ((or (atom tmp0)
           (progn (setq tmp1 (car tmp0)) nil)
           (progn
             (progn
              (setq x (car tmp1))
              (setq varl (cdr tmp1))
              tmp1)
             nil))
            nil)
   (seq
    (exit
     (do ((tmp2 varl (cdr tmp2)) (p nil))
```

```
        ((or (atom tmp2) (progn (setq p (car tmp2)) nil)) nil)
      (seq
       (exit
        (progn
         (setq prop (car p))
         (setq val (cdr p))
         (when val
           (progn
             (when (null (eq x '%))
               (|recordOldValue| x prop val))
             (when |$HiFiAccess|
               (|recordNewValue| x prop val))
             (rplacd p nil)))))))))
  (do ((i 1 (1+ i)))
      ((> i n) nil)
    (setq vec
      (unwind-protect (cdr (|readHiFi| i)) (|disableHist|)))
    (do ((tmp3 vec (cdr tmp3)) (p1 nil))
        ((or (atom tmp3) (progn (setq p1 (car tmp3)) nil)) nil)
     (setq x (car p1))
     (do ((tmp4 (cdr p1) (cdr tmp4)) (p2 nil))
         ((or (atom tmp4) (progn (setq p2 (car tmp4)) nil)) nil)
      (setq |$InteractiveFrame|
         (|putHist| x (car p2) (CDR p2) |$InteractiveFrame|)))))
  (setq val
   (and
    (setq p (assq '% vec))
    (setq p1 (assq '|value| (cdr p)))
    (cdr p1)))
  (setq |$InteractiveFrame| (|putHist| '% '|value| val |$InteractiveFrame|))
  (|updateHist|)))
```

———-

35.4.24 defun saveHistory

[sayKeyedMsg p327]
[makeInputFilename p973]
[histFileName p576]
[throwKeyedMsg p**??**]
[makeHistFileName p575]
[histInputFileName p576]
[writeInputLines p583]
[histFileErase p613]
[rdefiostream p**??**]
[spadrwrite0 p600]
[object2Identifier p**??**]

[rshut p??]
[namestring p1034]
[$seen p??]
[$HiFiAccess p724]
[$useInternalHistoryTable p575]
[$internalHistoryTable p??]

— defun saveHistory —

```
(defun |saveHistory| (fn)
 (let (|$seen| savefile inputfile saveStr n rec val)
 (declare (special |$seen| |$HiFiAccess| |$useInternalHistoryTable|
     |$internalHistoryTable|))
 (setq |$seen| (make-hash-table :test #'eq))
 (cond
  ((null |$HiFiAccess|)
    (|sayKeyedMsg| 's2ih0016 nil)) ; the history file is not on
  ((and (null |$useInternalHistoryTable|)
        (null (makeInputFilename (|histFileName|))))
    (|sayKeyedMsg| 's2ih0022 nil)) ; no history saved yet
  ((null fn)
    (|throwKeyedMsg| 's2ih0037 nil)) ; need to specify a history filename
  (t
    (setq savefile (|makeHistFileName| fn))
    (setq inputfile (|histInputFileName| fn))
    (|writeInputLines| fn 1)
    (|histFileErase| savefile)
    (when |$useInternalHistoryTable|
      (setq saveStr
        (rdefiostream
         (cons '(mode . output)
           (cons (cons 'file savefile) nil))))
      (do ((tmp0 (reverse |$internalHistoryTable|) (cdr tmp0))
           (tmp1 nil))
          ((or (atom tmp0)
               (progn (setq tmp1 (car tmp0)) nil)
               (progn
                (progn
                 (setq n (car tmp1))
                 (setq rec (cdr tmp1))
                 tmp1)
                nil))
               nil)
        (setq val (spadrwrite0 (|object2Identifier| n) rec saveStr))
        (when (eq val '|writifyFailed|)
         (|sayKeyedMsg| 's2ih0035 ; can't save the value of step
          (list n inputfile))))
      (rshut saveStr))
    (|sayKeyedMsg| 's2ih0018 ; saved history file is
```

```
      (cons (|namestring| savefile) nil))
    nil))))
```

———-

35.4.25 defun restoreHistory

[qcdr p??]
[qcar p??]
[identp p1040]
[throwKeyedMsg p??]
[makeHistFileName p575]
[putHist p586]
[makeInputFilename p973]
[sayKeyedMsg p327]
[namestring p1034]
[clearSpad2Cmd p496]
[histFileName p576]
[histFileErase p613]
[$fcopy p??]
[rkeyids p??]
[readHiFi p597]
[disableHist p599]
[updateInCoreHist p586]
[get p??]
[rempropI p??]
[clearCmdSortedCaches p497]
[$options p??]
[$internalHistoryTable p??]
[$HiFiAccess p724]
[$e p??]
[$useInternalHistoryTable p575]
[$InteractiveFrame p??]
[$oldHistoryFileName p574]

— **defun restoreHistory** —

```
(defun |restoreHistory| (fn)
 (let (|$options| fnq restfile curfile l oldInternal vec line x a)
 (declare (special |$options| |$internalHistoryTable| |$HiFiAccess| |$e|
     |$useInternalHistoryTable| |$InteractiveFrame| |$oldHistoryFileName|))
 (cond
   ((null fn) (setq fnq |$oldHistoryFileName|))
   ((and (consp fn)
         (eq (qcdr fn) nil)
```

```
        (progn
         (setq fnq (qcar fn))
         t)
        (identp fnq))
    (setq fnq fnq))
  (t (|throwKeyedMsg| 's2ih0023 (cons fnq nil)))) ; invalid filename
(setq restfile (|makeHistFileName| fnq))
(if (null (makeInputFilename restfile))
 (|sayKeyedMsg| 's2ih0024  ; file does not exist
      (cons (|namestring| restfile) nil))
 (progn
   (setq |$options| nil)
   (|clearSpad2Cmd| '(|all|))
   (setq curfile (|histFileName|))
   (|histFileErase| curfile)
   ($fcopy restfile curfile)
   (setq l (length (rkeyids curfile)))
   (setq |$HiFiAccess| t)
   (setq oldInternal |$useInternalHistoryTable|)
   (setq |$useInternalHistoryTable| nil)
   (when oldInternal (setq |$internalHistoryTable| nil))
   (do ((i 1 (1+ i)))
       ((> i l) nil)
     (setq vec (unwind-protect (|readHiFi| i) (|disableHist|)))
     (when oldInternal
       (setq |$internalHistoryTable|
        (cons (cons i vec) |$internalHistoryTable|)))
     (setq line (car vec))
     (dolist (p1 (cdr vec))
       (setq x (car p1))
       (do ((tmp1 (cdr p1) (cdr tmp1)) (p2 nil))
           ((or (atom tmp1) (progn (setq p2 (car tmp1)) nil)) nil)
        (setq |$InteractiveFrame|
          (|putHist| x
           (car p2) (cdr p2) |$InteractiveFrame|))))
     (|updateInCoreHist|))
   (setq |$e| |$InteractiveFrame|)
   (do ((tmp2 (caar |$InteractiveFrame|) (cdr tmp2)) (tmp3 nil))
       ((or (atom tmp2)
            (progn
              (setq tmp3 (car tmp2))
              nil)
            (progn
             (progn
               (setq a (car tmp3))
               tmp3)
               nil))
            nil)
      (when (|get| a '|localModemap| |$InteractiveFrame|)
        (|rempropI| a '|localModemap|)
```

```
          (|rempropI| a '|localVars|)
          (|rempropI| a '|mapBody|)))
    (setq |$IOindex| (1+ l))
    (setq |$useInternalHistoryTable| oldInternal)
    (|sayKeyedMsg| 'S2IH0025 ; workspace restored
      (cons (|namestring| restfile) nil))
    (|clearCmdSortedCaches|)
     nil))))
```

———-

35.4.26 defun setIOindex

[$IOindex p**??**]

— **defun setIOindex** —

```
(defun |setIOindex| (n)
  (declare (special |$IOindex|))
  (setq |$IOindex| n))
```

———-

35.4.27 defun showInput

[tab p**??**]
[readHiFi p597]
[disableHist p599]
[sayMSG p329]

— **defun showInput** —

```
(defun |showInput| (mini maxi)
 (let (vec l)
  (do ((|ind| mini (+ |ind| 1)))
      ((> |ind| maxi) nil)
    (setq vec (unwind-protect (|readHiFi| |ind|) (|disableHist|)))
    (cond
     ((> 10 |ind|) (tab 2))
     ((> 100 |ind|) (tab 1))
     (t nil))
    (setq l (car vec))
    (if (stringp l)
      (|sayMSG| (list "   [" |ind| "] " (car vec)))
      (progn
```

```
       (|sayMSG| (list "   [" |ind| "] "))
       (do ((tmp0 l (cdr tmp0)) (ln nil))
           ((or (atom tmp0) (progn (setq ln (car tmp0)) nil)) nil)
         (|sayMSG| (list "      " ln))))))))
```

———-

35.4.28 defun showInOut

[assq p1044]
[spadPrint p**??**]
[objValUnwrap p**??**]
[objMode p**??**]
[readHiFi p597]
[disableHist p599]
[sayMSG p329]

— **defun showInOut** —

```
(defun |showInOut| (mini maxi)
 (let (vec Alist triple)
  (do ((ind mini (+ ind 1)))
      ((> ind maxi) nil)
   (setq vec (unwind-protect (|readHiFi| ind) (|disableHist|)))
   (|sayMSG| (cons (car vec) nil))
   (cond
    ((setq Alist (assq '% (cdr vec)))
      (setq triple (cdr (assq '|value| (cdr Alist))))
      (setq |$IOindex| ind)
      (|spadPrint| (|objValUnwrap| triple) (|objMode| triple)))))))
```

———-

35.4.29 defun fetchOutput

[boot-equal p**??**]
[spaddifference p**??**]
[getI p**??**]
[throwKeyedMsg p**??**]
[readHiFi p597]
[disableHist p599]
[assq p1044]

— **defun fetchOutput** —

```
(defun |fetchOutput| (n)
 (let (vec Alist val)
  (cond
   ((and (boot-equal n (spaddifference 1)) (setq val (|getI| '% '|value|)))
     val)
   (|$HiFiAccess|
    (setq n
     (cond
      ((minusp n) (+ |$IOindex| n))
      (t n)))
    (cond
     ((>= n |$IOindex|)
       (|throwKeyedMsg| 'S2IH0001 (cons n nil))) ; no step n yet
     ((> 1 n)
       (|throwKeyedMsg| 's2ih0002 (cons n nil))) ; only nonzero steps
     (t
       (setq vec (unwind-protect (|readHiFi| n) (|disableHist|)))
       (cond
        ((setq Alist (assq '% (cdr vec)))
          (cond
            ((setq val (cdr (assq '|value| (cdr Alist))))
              val)
            (t
              (|throwKeyedMsg| 's2ih0003 (cons n nil))))) ; no step value
        (t (|throwKeyedMsg| 's2ih0003 (cons n nil))))))) ; no step value
   (t (|throwKeyedMsg| 's2ih0004 nil))))) ; history not on
```

———-

35.4.30 Read the history file using index n

[assoc p**??**]
[keyedSystemError p**??**]
[qcdr p**??**]
[rdefiostream p**??**]
[histFileName p576]
[spadrread p601]
[object2Identifier p**??**]
[rshut p**??**]
[$useInternalHistoryTable p575]
[$internalHistoryTable p**??**]

— defun readHiFi —

```
(defun |readHiFi| (n)
 "Read the history file using index n"
 (let (pair HiFi vec)
```

```
  (declare (special |$useInternalHistoryTable| |$internalHistoryTable|))
  (if |$useInternalHistoryTable|
   (progn
    (setq pair (|assoc| n |$internalHistoryTable|))
    (if (atom pair)
     (|keyedSystemError| 's2ih0034 nil) ; missing element
     (setq vec (qcdr pair))))
   (progn
     (setq HiFi
      (rdefiostream
       (cons
        '(mode . input)
         (cons
          (cons 'file (|histFileName|)) nil))))
     (setq vec (spadrread (|object2Identifier| n) HiFi))
     (rshut HiFi)))
  vec))
```

———-

35.4.31 Write information of the current step to history file

[rdefiostream p??]
[histFileName p576]
[spadrwrite p601]
[object2Identifier p??]
[rshut p??]
[$useInternalHistoryTable p575]
[$internalHistoryTable p??]
[$IOindex p??]
[$HistRecord p??]
[$currentLine p??]

— **defun writeHiFi** —

```
(defun |writeHiFi| ()
 "Writes information of the current step to history file"
 (let (HiFi)
 (declare (special |$useInternalHistoryTable| |$internalHistoryTable|
     |$IOindex| |$HistRecord| |$currentLine|))
   (if |$useInternalHistoryTable|
     (setq |$internalHistoryTable|
      (cons
       (cons |$IOindex|
        (cons |$currentLine| |$HistRecord|))
        |$internalHistoryTable|))
    (progn
```

```
      (setq HiFi
        (rdefiostream
         (cons
           '(mode . output)
            (cons (cons 'file (|histFileName|)) nil))))
      (spadrwrite (|object2Identifier| |$IOindex|)
        (cons |$currentLine| |$HistRecord|) HiFi)
      (rshut HiFi)))))
```

———-

35.4.32 Disable history if an error occurred

[histFileErase p613]
[histFileName p576]
[$HiFiAccess p724]

— **defun disableHist** —

```
(defun |disableHist| ()
 "Disable history if an error occurred"
 (declare (special |$HiFiAccess|))
 (cond
  ((null |$HiFiAccess|)
    (|histFileErase| (|histFileName|)))
  (t nil)))
```

———-

35.4.33 defun writeHistModesAndValues

[get p??]
[putHist p586]
[$InteractiveFrame p??]

— **defun writeHistModesAndValues** —

```
(defun |writeHistModesAndValues| ()
 (let (a x)
 (declare (special |$InteractiveFrame|))
  (do ((tmp0 (caar |$InteractiveFrame|) (cdr tmp0)) (tmp1 nil))
      ((or (atom tmp0)
           (progn
             (setq tmp1 (car tmp0))
             nil)
```

```
          (progn
            (progn
              (setq a (car tmp1))
              tmp1)
            nil))
        nil)
      (cond
       ((setq x (|get| a '|value| |$InteractiveFrame|))
        (|putHist| a '|value| x |$InteractiveFrame|))
       ((setq x (|get| a '|mode| |$InteractiveFrame|))
        (|putHist| a '|mode| x |$InteractiveFrame|))))))
```

———-

35.5 Lisplib output transformations

Lisplib output transformations

Some types of objects cannot be saved by LISP/VM in lisplibs. These functions transform an object to a writable form and back.

35.5.1 defun spadrwrite0

[safeWritify p602]
[rwrite p600]

— **defun spadrwrite0** —

```
(defun spadrwrite0 (vec item stream)
 (let (val)
  (setq val (|safeWritify| item))
  (if (eq val '|writifyFailed|)
    val
    (progn
     (|rwrite| vec val stream)
     item))))
```

———-

35.5.2 defun Random write to a stream

[rwrite p600]
[pname p1039]
[identp p1040]

— **defun rwrite** —

```
(defun |rwrite| (key val stream)
  (when (identp key) (setq key (pname key)))
  (rwrite key val stream)))
```

————-

35.5.3 defun spadrwrite

[spadrwrite0 p600]
[throwKeyedMsg p**??**]

— **defun spadrwrite** —

```
(defun spadrwrite (vec item stream)
 (let (val)
  (setq val (spadrwrite0 vec item stream))
  (if (eq val '|writifyFailed|)
   (|throwKeyedMsg| 's2ih0036 nil) ; cannot save value to file
   item)))
```

————-

35.5.4 defun spadrread

[dewritify p611]
[rread p601]

— **defun spadrread** —

```
(defun spadrread (vec stream)
 (|dewritify| (|rread| vec stream nil)))
```

————-

35.5.5 defun Random read a key from a stream

RREAD takes erroval to return if key is missing [rread p601]
[identp p1040]
[pname p1039]

— **defun rread** —

```
(defun |rread| (key rstream errorval)
  (when (identp key) (setq key (pname key)))
  (rread key rstream errorval))
```

———-

35.5.6 defun unwritable?

[vecp p**??**]
[placep p**??**]

— **defun unwritable?** —

```
(defun |unwritable?| (ob)
 (cond
  ((or (consp ob) (vecp ob)) nil)
  ((or (compiled-function-p ob) (hash-table-p ob)) t)
  ((or (placep ob) (readtablep ob)) t)
  ((floatp ob) t)
  (t nil)))
```

———-

35.5.7 defun writifyComplain

Create a full isomorphic object which can be saved in a lisplib. Note that `dewritify(writify(x))` preserves UEQUALity of hashtables. HASHTABLEs go both ways. READTABLEs cannot presently be transformed back. [sayKeyedMsg p327]
[$writifyComplained p**??**]

— **defun writifyComplain** —

```
(defun |writifyComplain| (s)
 (declare (special |$writifyComplained|))
 (unless |$writifyComplained|
  (setq |$writifyComplained| t)
  (|sayKeyedMsg| 's2ih0027 (list s)))) ; cannot save value
```

———-

35.5.8 defun safeWritify

[writifyTag p**??**]
[writify p606]

— **defun safeWritify** —

```
(defun |safeWritify| (ob)
 (catch '|writifyTag| (|writify| ob)))
```

————-

35.5.9 defun writify,writifyInner

[writifyTag p**??**]
[seq p**??**]
[exit p**??**]
[hget p1038]
[qcar p**??**]
[qcdr p**??**]
[spadClosure? p607]
[writify,writifyInner p603]
[hput p1038]
[qrplaca p**??**]
[qrplacd p**??**]
[vecp p**??**]
[isDomainOrPackage p865]
[mkEvalable p903]
[devaluate p**??**]
[qvmaxindex p**??**]
[qsetvelt p**??**]
[qvelt p**??**]
[constructor? p**??**]
[hkeys p1038]
[hashtable-class p**??**]
[placep p**??**]
[boot-equal p**??**]
[$seen p**??**]
[$NonNullStream p607]
[$NullStream p607]

— **defun writify,writifyInner** —

```
(defun |writify,writifyInner| (ob)
 (prog (e name tmp1 tmp2 tmp3 x qcar qcdr d n keys nob)
  (declare (special |$seen| |$NonNullStream| |$NullStream|))
  (return
   (seq
    (when (null ob) (exit nil))
```

```
(when (setq e (hget |$seen| ob)) (exit e))
(when (consp ob)
 (exit
  (seq
   (setq qcar (qcar ob))
   (setq qcdr (qcdr ob))
   (when (setq name (|spadClosure?| ob))
    (exit
     (seq
      (setq d (|writify,writifyInner| (qcdr ob)))
      (setq nob
       (cons 'writified!!
        (cons 'spadclosure
         (cons d (cons name nil)))))
      (hput |$seen| ob nob)
      (hput |$seen| nob nob)
      (exit nob))))
   (when
    (and
     (and (consp ob)
          (eq (qcar ob) 'lambda-closure)
          (progn
           (setq tmp1 (qcdr ob))
           (and (consp tmp1)
                (progn
                 (setq tmp2 (qcdr tmp1))
                 (and
                  (consp tmp2)
                  (progn
                   (setq tmp3 (qcdr tmp2))
                   (and (consp tmp3)
                        (progn
                         (setq x (qcar tmp3))
                         t))))))))) x)
    (exit
     (throw '|writifyTag| '|writifyFailed|)))
   (setq nob (cons qcar qcdr))
   (hput |$seen| ob nob)
   (hput |$seen| nob nob)
   (setq qcar (|writify,writifyInner| qcar))
   (setq qcdr (|writify,writifyInner| qcdr))
   (qrplaca nob qcar)
   (qrplacd nob qcdr)
   (exit nob))))
(when (vecp ob)
 (exit
  (seq
   (when (|isDomainOrPackage| ob)
     (setq d (|mkEvalable| (|devaluate| ob)))
     (setq nob (list 'writified!! 'devaluated (|writify,writifyInner| d)))
```

```
      (hput |$seen| ob nob)
      (hput |$seen| nob nob)
      (exit nob))
    (setq n (qvmaxindex ob))
    (setq nob (make-array (1+ n)))
    (hput |$seen| ob nob)
    (hput |$seen| nob nob)
    (do ((i 0 (=! i)))
        ((> i n) nil)
      (qsetvelt nob i (|writify,writifyInner| (qvelt ob i))))
    (exit nob))))
(when (eq ob 'writified!!)
 (exit
  (cons 'writified!! (cons 'self nil))))
(when (|constructor?| ob)
 (exit ob))
(when (compiled-function-p ob)
 (exit
  (throw '|writifyTag| '|writifyFailed|)))
(when (hash-table-p ob)
   (setq nob (cons 'writified!! nil))
   (hput |$seen| ob nob)
   (hput |$seen| nob nob)
   (setq keys (hkeys ob))
   (qrplacd nob
    (cons
      'hashtable
      (cons
        (hashtable-class ob)
        (cons
          (|writify,writifyInner| keys)
          (cons
            (prog (tmp0)
              (setq tmp0 nil)
              (return
               (do ((tmp1 keys (cdr tmp1)) (k nil))
                   ((or (atom tmp1)
                        (progn
                         (setq k (car tmp1))
                          nil))
                        (nreverse0 tmp0))
                 (setq tmp0
                  (cons (|writify,writifyInner| (hget ob k)) tmp0)))))
             nil)))))
   (exit nob))
(when (placep ob)
   (setq nob (cons 'writified!! (cons 'place nil)))
   (hput |$seen| ob nob)
   (hput |$seen| nob nob)
   (exit nob))
```

```
    (when (readtablep ob)
     (exit
      (throw '|writifyTag| '|writifyFailed|)))
    (when (stringp ob)
     (exit
      (seq
       (when (eq ob |$NullStream|)
        (exit
         (cons 'writified!! (cons 'nullstream nil))))
       (when (eq ob |$NonNullStream|)
        (exit
         (cons 'writified!! (cons 'nonnullstream nil))))
       (exit ob))))
    (when (floatp ob)
     (exit
      (seq
       (when (boot-equal ob (read-from-string (princ-to-string ob)))
        (exit ob))
       (exit
        (cons 'writified!!
         (cons 'float
          (cons ob
           (multiple-value-list (integer-decode-float ob)))))))))
    (exit ob)))))
```

———-

35.5.10 defun writify

[ScanOrPairVec p612]
[function p**??**]
[writify,writifyInner p603]
[$seen p**??**]
[$writifyComplained p**??**]

— defun writify —

```
(defun |writify| (ob)
 (let (|$seen| |$writifyComplained|)
 (declare (special |$seen| |$writifyComplained|))
  (if (null (|ScanOrPairVec| (|function| |unwritable?|) ob))
    ob
    (progn
     (setq |$seen| (make-hash-table :test #'eq))
     (setq |$writifyComplained| nil)
     (|writify,writifyInner| ob)))))
```

———-

35.5.11 defun spadClosure?

[qcar p??]
[bpiname p??]
[qcdr p??]
[vecp p??]

— **defun spadClosure?** —

```
(defun |spadClosure?| (ob)
 (let (fun name vec)
  (setq fun (qcar ob))
  (if (null (setq name (bpiname fun)))
   nil
   (progn
     (setq vec (qcdr ob))
     (if (null (vecp vec))
      nil
      name)))))
```

———-

35.5.12 defvar $NonNullStream

— **initvars** —

```
(defvar |$NonNullStream| "NonNullStream")
```

———-

35.5.13 defvar $NullStream

— **initvars** —

```
(defvar |$NullStream| "NullStream")
```

———-

35.5.14 defun dewritify,dewritifyInner

[seq p??]
[exit p??]
[hget p1038]
[intp p??]
[gensymmer p??]
[error p??]
[poundsign p??]
[hput p1038]
[dewritify,dewritifyInner p608]
[concat p1041]
[vmread p??]
[make-instream p971]
[spaddifference p??]
[qcar p??]
[qcdr p??]
[qrplaca p??]
[qrplacd p??]
[vecp p??]
[qvmaxindex p??]
[qsetvelt p??]
[qvelt p??]
[$seen p??]
[$NullStream p607]
[$NonNullStream p607]

— defun dewritify,dewritifyInner —

```
(defun |dewritify,dewritifyInner| (ob)
 (prog (e type oname f vec name tmp1 signif expon sign fval qcar qcdr n nob)
  (declare (special |$seen| |$NullStream| |$NonNullStream|))
  (return
   (seq
    (when (null ob)
      (exit nil))
    (when (setq e (hget |$seen| ob))
      (exit e))
    (when (and (consp ob) (eq (car ob) 'writified!!))
     (exit
      (seq
       (setq type (elt ob 1))
       (when (eq type 'self)
        (exit 'writified!!))
       (when (eq type 'bpi)
        (exit
         (seq
```

```
   (setq oname (elt ob 2))
   (setq f
    (seq
     (when (integerp oname) (exit (eval (gensymmer oname))))
     (exit (symbol-function oname))))
   (when (null (compiled-function-p f))
    (exit (|error| "A required BPI does not exist.")))
   (when (and (> (|#| ob) 3) (not (equal (sxhash f) (elt ob 3))))
    (exit (|error| "A required BPI has been redefined.")))
   (hput |$seen| ob f)
   (exit f))))
(when (eq type 'hashtable)
 (exit
  (seq
   (setq nob (make-hash-table :test #'equal))
   (hput |$seen| ob nob)
   (hput |$seen| nob nob)
   (do ((tmp0 (elt ob 3) (cdr tmp0))
        (k nil)
        (tmp1 (elt ob 4) (cdr tmp1))
        (e nil))
       ((or (atom tmp0)
            (progn
             (setq k (car tmp0))
             nil)
            (atom tmp1)
            (progn
             (setq e (car tmp1))
             nil))
           nil)
    (seq
     (exit
      (hput nob (|dewritify,dewritifyInner| k)
       (|dewritify,dewritifyInner| e)))))
   (exit nob))))
(when (eq type 'devaluated)
 (exit
  (seq
   (setq nob (eval (|dewritify,dewritifyInner| (elt ob 2))))
   (hput |$seen| ob nob)
   (hput |$seen| nob nob)
   (exit nob))))
(when (eq type 'spadclosure)
 (exit
  (seq
   (setq vec (|dewritify,dewritifyInner| (elt ob 2)))
   (setq name (ELT ob 3))
   (when (null (fboundp name))
    (exit
     (|error|
```

```
          (concat "undefined function: " (symbol-name name)))))
       (setq nob (cons (symbol-function name) vec))
       (hput |$seen| ob nob)
       (hput |$seen| nob nob)
       (exit nob))))
   (when (eq type 'place)
    (exit
     (seq
      (setq nob (vmread (make-instream nil)))
      (hput |$seen| ob nob)
      (hput |$seen| nob nob)
      (exit nob))))
   (when (eq type 'readtable)
    (exit (|error| "Cannot de-writify a read table.")))
   (when (eq type 'nullstream)
    (exit |$NullStream|))
   (when (eq type 'nonnullstream)
    (exit |$NonNullStream|))
   (when (eq type 'float)
    (exit
     (seq
      (progn
       (setq tmp1 (cddr ob))
       (setq fval (car tmp1))
       (setq signif (cadr tmp1))
       (setq expon (caddr tmp1))
       (setq sign (cadddr tmp1))
       tmp1)
      (setq fval (scale-float (float signif fval) expon))
      (when (minusp sign)
       (exit (spaddifference fval)))
      (exit fval))))
   (exit (|error| "Unknown type to de-writify."))))
(when (consp ob)
 (exit
  (seq
   (setq qcar (qcar ob))
   (setq qcdr (qcdr ob))
   (setq nob (cons qcar qcdr))
   (hput |$seen| ob nob)
   (hput |$seen| nob nob)
   (qrplaca nob (|dewritify,dewritifyInner| qcar))
   (qrplacd nob (|dewritify,dewritifyInner| qcdr))
   (exit nob))))
(when (vecp ob)
 (exit
  (seq
   (setq n (qvmaxindex ob))
   (setq nob (make-array (1+ n)))
   (hput |$seen| ob nob)
```

```
       (hput |$seen| nob nob)
       (do ((i 0 (1+ i)))
           ((> i n) nil)
        (seq
         (exit
          (qsetvelt nob i
            (|dewritify,dewritifyInner| (qvelt ob i))))))
       (exit nob))))
    (exit ob)))))
```

———-

35.5.15 defun dewritify

[ScanOrPairVec p612]
[function p**??**]
[dewritify,dewritifyInner p608]
[$seen p**??**]

— **defun dewritify** —

```
(defun |dewritify| (ob)
 (let (|$seen|)
 (declare (special |$seen|))
   (if (null (|ScanOrPairVec| #'(lambda (a) (eq a 'writified!!)) ob))
    ob
    (progn
      (setq |$seen| (make-hash-table :test #'eq))
      (|dewritify,dewritifyInner| ob)))))
```

———-

35.5.16 defun ScanOrPairVec,ScanOrInner

[ScanOrPairVecAnswer p**??**]
[hget p1038]
[hput p1038]
[ScanOrPairVec,ScanOrInner p611]
[qcar p**??**]
[qcdr p**??**]
[vecp p**??**]
[$seen p**??**]

— **defun ScanOrPairVec,ScanOrInner** —

```
(defun |ScanOrPairVec,ScanOrInner| (f ob)
  (declare (special |$seen|))
  (when (hget |$seen| ob) nil)
  (when (consp ob)
    (hput |$seen| ob t)
    (|ScanOrPairVec,ScanOrInner| f (qcar ob))
    (|ScanOrPairVec,ScanOrInner| f (qcdr ob)))
  (when (vecp ob)
     (hput |$seen| ob t)
     (do ((tmp0 (spaddifference (|#| ob) 1)) (i 0 (1+ i)))
         ((> i tmp0) nil)
       (|ScanOrPairVec,ScanOrInner| f (elt ob i))))
  (when (funcall f ob) (throw '|ScanOrPairVecAnswer| t))
  nil)
```

———-

35.5.17 defun ScanOrPairVec

[ScanOrPairVecAnswer p**??**]
[ScanOrPairVec,ScanOrInner p611]
[$seen p**??**]

— **defun ScanOrPairVec** —

```
(defun |ScanOrPairVec| (f ob)
 (let (|$seen|)
 (declare (special |$seen|))
  (setq |$seen| (make-hash-table :test #'eq))
  (catch '|ScanOrPairVecAnswer| (|ScanOrPairVec,ScanOrInner| f ob))))
```

———-

35.5.18 defun gensymInt

[gensymp p**??**]
[error p**??**]
[pname p1039]
[charDigitVal p613]

— **defun gensymInt** —

```
(defun |gensymInt| (g)
 (let (p n)
  (if (null (gensymp g))
```

```
   (|error| "Need a GENSYM")
   (progn
     (setq p (pname g))
     (setq n 0)
     (do ((tmp0 (spaddifference (|#| p) 1)) (i 2 (1+ i)))
         ((> i tmp0) nil)
       (setq n (+ (* 10 n) (|charDigitVal| (elt p i)))))
     n))))
```

———-

35.5.19 defun charDigitVal

[spaddifference p**??**]
[error p**??**]

— **defun charDigitVal** —

```
(defun |charDigitVal| (c)
 (let (digits n)
  (setq digits "0123456789")
  (setq n (spaddifference 1))
  (do ((tmp0 (spaddifference (|#| digits) 1)) (i 0 (1+ i)))
      ((or (> i tmp0) (null (minusp n))) nil)
   (if (char= c (elt digits i))
    (setq n i)
    nil))
  (if (minusp n)
   (|error| "Character is not a digit")
   n)))
```

———-

35.5.20 defun histFileErase

— **defun histFileErase** —

```
(defun |histFileErase| (file)
 (when (probe-file file) (delete-file file)))
```

———-

35.6 History File Messages

— History File Messages —

```
S2IH0001
 You have not reached step %1b yet, and so its value cannot be
 supplied.
S2IH0002
 Cannot supply value for step %1b because 1 is the first step.
S2IH0003
 Step %1b has no value.
S2IH0004
 The history facility is not on, so you cannot use %b %% %d .
S2IH0006
 You have not used the correct syntax for the %b history %d command.
 Issue %b )help history %d for more information.
S2IH0007
 The history facility is already on.
S2IH0008
 The history facility is now on.
S2IH0009
 Turning on the history facility will clear the contents of the
 workspace.
 Please enter %b y %d or %b yes %d if you really want to do this:
S2IH0010
 The history facility is still off.
S2IH0011
 The history facility is already off.
S2IH0012
 The history facility is now off.
S2IH0013
 The history facility is not on, so the .input file containing your user input
 cannot be created.
S2IH0014
 Edit %b %1 %d to see the saved input lines.
S2IH0015
 The argument %b n %d for %b )history )change n must be a nonnegative
 integer and your argument, %1b , is not one.
S2IH0016
 The history facility is not on, so no information can be saved.
S2IH0018
 The saved history file is %1b .
S2IH0019
 There is no history file, so value of step %1b is
 undefined.
S2IH0022
 No history information had been saved yet.
S2IH0023
```

```
 %1b is not a valid filename for the history file.
S2IH0024
 History information cannot be restored from %1b because the file does
 not exist.
S2IH0025
 The workspace has been successfully restored from the history file
 %1b .
S2IH0026
 The history facility command %1b cannot be performed because the
 history facility is not on.
S2IH0027
 A value containing a %1b is being saved in a history file or a
 compiled input file INLIB. This type
 is not yet usable in other history operations.  You might want to issue
 %b )history )off %d
S2IH0029
 History information is already being maintained in an external file
 (and not in memory).
S2IH0030
 History information is already being maintained in memory (and not
 in an external file).
S2IH0031
 When the history facility is active, history information will be
 maintained in a file (and not in an internal table).
S2IH0032
 When the history facility is active, history information will be
 maintained in memory (and not in an external file).
S2IH0034
 Missing element in internal history table.
S2IH0035
 Can't save the value of step number %1b.  You can re-generate this value
 by running the input file %2b.
S2IH0036
 The value specified cannot be saved to a file.
S2IH0037
 You must specify a file name to the history save command
S2IH0038
 You must specify a file name to the history write command
```

Chapter 36

)include help page Command

36.1 include help page man page

— **include.help** —

```
User Level Required: interpreter

Command Syntax:

 )include filename

Command Description:

The )include command can be used in .input files to place the contents
of another file inline with the current file.  The path can be an
absolute or relative pathname.
```

————-

36.2 Functions

36.2.1 defun ncloopInclude1

[ncloopIncFileName p618]
[ncloopInclude p618]

— **defun ncloopInclude1** —

```
(defun |ncloopInclude1| (name n)
 (let (a)
  (if (setq a (|ncloopIncFileName| name))
    (|ncloopInclude| a n)
    n)))
```

———-

36.2.2 Returns the first non-blank substring of the given string

[incFileName p618]
[concat p1041]

— **defun ncloopIncFileName** —

```
(defun |ncloopIncFileName| (string)
 "Returns the first non-blank substring of the given string"
 (let (fn)
  (unless (setq fn (|incFileName| string))
   (write-line (concat string " not found")))
 fn))
```

———-

36.2.3 Open the include file and read it in

The ncloopInclude0 function is part of the parser and lives in int-top.boot. [ncloopInclude0 p73]

— **defun ncloopInclude** —

```
(defun |ncloopInclude| (name n)
 "Open the include file and read it in"
 (with-open-file (st name) (|ncloopInclude0| st name n)))
```

———-

36.2.4 Return the include filename

Given a string we return the first token from the string which is the first non-blank substring. [incBiteOff p619]

— **defun incFileName** —

```
(defun |incFileName| (x)
 "Return the include filename"
  (car (|incBiteOff| x)))
```

———-

36.2.5 Return the next token

Takes a sequence and returns the a list of the first token and the remaining string characters. If there are no remaining string characters the second string is of length 0. Effectively it "bites off" the first token in the string. If the string only 0 or more blanks it returns nil.

— **defun incBiteOff** —

```
(defun |incBiteOff| (x)
 "Return the next token"
 (let (blank nonblank)
  (setq x (string x))
  (when (setq nonblank (position #\space x :test-not #'char=))
    (setq blank (position #\space x :start nonblank))
    (if blank
     (list (subseq x nonblank blank) (subseq x blank))
     (list (subseq x nonblank) "")))))
```

———-

Chapter 37

)library help page Command

37.1 library help page man page

— library.help —

```
=======================================================================
A.14.  )library
=======================================================================

User Level Required:  interpreter

Command Syntax:

  - )library libName1  [libName2 ...]
  - )library )dir dirName
  - )library )only objName1  [objlib2 ...]
  - )library )noexpose

Command Description:

This command replaces the )load system command that was available in AXIOM
releases before version 2.0. The )library command makes available to AXIOM
the compiled objects in the libraries listed.

For example, if you )compile dopler.spad in your home directory, issue )library
dopler to have AXIOM look at the library, determine the category and domain
constructors present, update the internal database with various properties of
the constructors, and arrange for the constructors to be automatically loaded
when needed. If the )noexpose option has not been given, the constructors
will be exposed (that is, available) in the current frame.

If you compiled a file you will have an NRLIB present, for example,
```

DOPLER.NRLIB, where DOPLER is a constructor abbreviation. The command)library DOPLER will then do the analysis and database updates as above.

To tell the system about all libraries in a directory, use)library)dir dirName where dirName is an explicit directory. You may specify ‘‘.’’ as the directory, which means the current directory from which you started the system or the one you set via the)cd command. The directory name is required.

You may only want to tell the system about particular constructors within a library. In this case, use the)only option. The command)library dopler)only Test1 will only cause the Test1 constructor to be analyzed, autoloaded, etc..

Finally, each constructor in a library are usually automatically exposed when the)library command is used. Use the)noexpose option if you not want them exposed. At a later time you can use)set expose add constructor to expose any hidden constructors.

Note for AXIOM beta testers: At various times this command was called)local and)with before the name)library became the official name.

Also See:
o)cd
o)compile
o)frame
o)set

———————

1

[1] “cd” (?? p ??) “frame” (33.5.16 p 561) “set” (45.36.1 p 799)

Chapter 38

)lisp help page Command

38.1 lisp help page man page

— lisp.help —

```
========================================================================
A.15.  )lisp
========================================================================

User Level Required:  development

Command Syntax:

  -  )lisp [lispExpression]

Command Description:

This command is used by AXIOM system developers to have single expressions
evaluated by the Lisp system on which AXIOM is built. The lispExpression is
read by the Lisp reader and evaluated. If this expression is not complete
(unbalanced parentheses, say), the reader will wait until a complete
expression is entered.

Since this command is only useful for evaluating single expressions, the )fin
command may be used to drop out of AXIOM into Lisp.

Also See:
o )system
o )boot
o )fin
```

[1]

38.2 Functions

This command is in the list of `$noParseCommands` 19.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 19.2.1

[1] "system" (?? p ??) "boot" (5.1.8 p 25) "fin" (32.1.1 p 544)

Chapter 39

)load help page Command

39.1 load help page man page

— **load.help** —

```
==========================================================================
A.16.  )load
==========================================================================

User Level Required:  interpreter

Command Description:

This command is obsolete. Use )library instead.
```

———-

39.1.1 defun The)load command (obsolete)

We keep this command around in case anyone has the original Axiom book. [sayKeyedMsg p327]

— **defun load** —

```
(defun |load| (ignore)
 (declare (ignore ignore))
 (|sayKeyedMsg| 'S2IU0003 nil))
```

———-

Chapter 40

)ltrace help page Command

40.1 ltrace help page man page

— **ltrace.help** —

```
========================================================================
A.17.  )ltrace
========================================================================

User Level Required:  development

Command Syntax:

This command has the same arguments as options as the )trace command.

Command Description:

This command is used by AXIOM system developers to trace Lisp or BOOT
functions. It is not supported for general use.

Also See:
o )boot
o )lisp
o )trace
```

———-

1

[1] "boot" (5.1.8 p 25) "lisp" (?? p ??) "trace" (51.1.7 p 837)

40.1.1 defun The top level)ltrace function

[trace p837]

— **defun ltrace** —

```
(defun |ltrace| (arg) (|trace| arg))
```

————-

40.2 Variables Used

40.3 Functions

Chapter 41

)pquit help page Command

41.1 pquit help page man page

— pquit.help —

```
====================================================================
A.18.  )pquit
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )pquit

Command Description:

This command is used to terminate AXIOM and return to the operating system.
Other than by redoing all your computations or by using the )history )restore
command to try to restore your working environment, you cannot return to
AXIOM in the same state.

)pquit differs from the )quit in that it always asks for confirmation that
you want to terminate AXIOM (the ‘‘p’’ is for ‘‘protected’’). When you enter
the )pquit command, AXIOM responds

      Please enter y or yes if you really want to leave the interactive
                environment and return to the operating system:

If you respond with y or yes, you will see the message

            You are now leaving the AXIOM interactive environment.
```

```
    Issue the command axiom to the operating system to start a new session.

and AXIOM will terminate and return you to the operating system (or the
environment from which you invoked the system). If you responded with
something other than y or yes, then the message

        You have chosen to remain in the AXIOM interactive environment.

will be displayed and, indeed, AXIOM would still be running.

Also See:
o )fin
o )history
o )close
o )quit
o )system
```

———-

[1]

41.2 Functions

41.2.1 The top level pquit command

[pquitSpad2Cmd p630]

— **defun pquit** —

```
(defun |pquit| ()
 "The top level pquit command"
 (|pquitSpad2Cmd|))
```

———-

41.2.2 The top level pquit command handler

[quitSpad2Cmd p634]
[$quitCommandType p791]

— **defun pquitSpad2Cmd** —

[1] "fin" (32.1.1 p 544) "history" (35.4.7 p 578) "close" (25.2.2 p 506) "quit" (42.2.1 p 634) "system" (?? p ??)

```
(defun |pquitSpad2Cmd| ()
 "The top level pquit command handler"
 (let ((|$quitCommandType| '|protected|))
  (declare (special |$quitCommandType|))
  (|quitSpad2Cmd|)))
```

———-

This command is in the list of `$noParseCommands` 19.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 19.2.1

Chapter 42

)quit help page Command

42.1 quit help page man page

— quit.help —

```
====================================================================
A.19.  )quit
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )quit
  - )set quit protected | unprotected

Command Description:

This command is used to terminate AXIOM and return to the operating system.
Other than by redoing all your computations or by using the )history )restore
command to try to restore your working environment, you cannot return to
AXIOM in the same state.

)quit differs from the )pquit in that it asks for confirmation only if the
command

)set quit protected

has been issued. Otherwise, )quit will make AXIOM terminate and return you to
the operating system (or the environment from which you invoked the system).

The default setting is )set quit protected so that )quit and )pquit behave in
```

```
the same way. If you do issue

)set quit unprotected

we suggest that you do not (somehow) assign )quit to be executed when you
press, say, a function key.

Also See:
o )fin
o )history
o )close
o )pquit
o )system
```

———-

[1]

42.2 Functions

42.2.1 The top level quit command

[quitSpad2Cmd p634]

— **defun quit** —

```
(defun |quit| ()
 "The top level quit command"
 (|quitSpad2Cmd|))
```

———-

42.2.2 The top level quit command handler

[upcase p**??**]
[queryUserKeyedMsg p**??**]
[string2id-n p**??**]
[leaveScratchpad p635]
[sayKeyedMsg p327]
[tersyscommand p448]
[$quitCommandType p791]

[1] "fin" (32.1.1 p 544) "history" (35.4.7 p 578) "close" (25.2.2 p 506) "pquit" (41.2.1 p 630) "system" (**??** p **??**)

— **defun quitSpad2Cmd** —

```
(defun |quitSpad2Cmd| ()
 "The top level quit command handler"
 (declare (special |$quitCommandType|))
 (if (eq |$quitCommandType| '|protected|)
  (let (x)
   (setq x (upcase (|queryUserKeyedMsg| 's2iz0031 nil)))
   (when (member (string2id-n x 1) '(y yes)) (|leaveScratchpad|))
   (|sayKeyedMsg| 's2iz0032 nil)
   (tersyscommand))
   (|leaveScratchpad|)))
```

———-

42.2.3 Leave the Axiom interpreter

— **defun leaveScratchpad** —

```
(defun |leaveScratchpad| ()
 "Leave the Axiom interpreter"
 (bye))
```

———-

This command is in the list of `$noParseCommands` 19.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 19.2.1

Chapter 43

)read help page Command

43.1 read help page man page

— read.help —

```
====================================================================
A.20.  )read
====================================================================

User Level Required:  interpreter

Command Syntax:

  -  )read [fileName]
  -  )read [fileName] []quiet] []ifthere]

Command Description:

This command is used to read .input files into AXIOM. The command

)read matrix.input

will read the contents of the file matrix.input into AXIOM. The ‘‘.input’’
file extension is optional. See the AXIOM User Guide index for more
information about .input files.

This command remembers the previous file you edited, read or compiled. If you
do not specify a file name, the previous file will be read.

The )ifthere option checks to see whether the .input file exists. If it does
not, the )read command does nothing. If you do not use this option and the
file does not exist, you are asked to give the name of an existing .input
```

```
file.

The )quiet option suppresses output while the file is being read.

Also See:
o )compile
o )edit
o )history
```

———-

[1]

43.1.1 defun The)read command

[readSpad2Cmd p638]

— **defun read** —

```
(defun |read| (arg) (|readSpad2Cmd| arg))
```

———-

43.1.2 defun Implement the)read command

[selectOptionLC p475]
[optionError p445]
[pathname p1036]
[pathnameTypeId p1035]
[makePathname p1036]
[pathnameName p1034]
[mergePathnames p1035]
[findfile p**??**]
[throwKeyedMsg p**??**]
[namestring p1034]
[upcase p**??**]
[member p1042]
[/read p640]
[$InteractiveMode p24]
[$findfile p**??**]
[$UserLevel p798]
[$options p**??**]
[/editfile p511]

[1] "edit" (31.2.1 p 540) "history" (35.4.7 p 578)

— defun readSpad2Cmd —

```
(defun |readSpad2Cmd| (arg)
 (prog (|$InteractiveMode| fullopt ifthere quiet ef devFTs fileTypes
        ll ft upft fs)
 (declare (special |$InteractiveMode| $findfile |$UserLevel| |$options|
                   /editfile))
  (setq |$InteractiveMode| t)
  (dolist (opt |$options|)
   (setq fullopt
    (|selectOptionLC| (caar opt) '(|quiet| |test| |ifthere|) '|optionError|))
    (cond
     ((eq fullopt '|ifthere|) (setq ifthere t))
     ((eq fullopt '|quiet|)   (setq quiet t))))
  (setq ef (|pathname| /editfile))
  (when (eq (|pathnameTypeId| ef) 'spad)
    (setq ef (|makePathname| (|pathnameName| ef) "*" "*")))
  (if arg
   (setq arg (|mergePathnames| (|pathname| arg) ef))
   (setq arg ef))
  (setq devFTs '("input" "INPUT" "boot" "BOOT" "lisp" "LISP"))
  (setq fileTypes
   (cond
    ((eq |$UserLevel| '|interpreter|) '("input" "INPUT"))
    ((eq |$UserLevel| '|compiler|)    '("input" "INPUT"))
    (t devFTs)))
  (setq ll ($findfile arg fileTypes))
  (unless ll
    (if ifthere
     (return nil)
     (|throwKeyedMsg| 'S2IL0003  (list (|namestring| arg)))))
  (setq ll (|pathname| ll))
  (setq ft (|pathnameType| ll))
  (setq upft (upcase ft))
  (cond
   ((null (|member| upft fileTypes))
     (setq fs (|namestring| arg))
     (if (|member| upft devFTs)
      (|throwKeyedMsg| 'S2IZ0033 (list fs))
      (|throwKeyedMsg| 'S2IZ0034 (list fs))))
   (t
    (setq /editfile ll)
    (when (string= upft "BOOT") (setq |$InteractiveMode| nil))
    (/read ll quiet)))))
```

————-

43.1.3 defun /read

[/read /rf (vol9)]
[/read /rq (vol9)]
[/editfile p511]

— **defun /read** —

```
(defun /read (l q)
 (declare (special /editfile))
  (setq /editfile l)
  (cond
    (q (/rq))
    (t (/rf)) )
  (flag |boot-NewKEY| 'key)
  (|terminateSystemCommand|)
  (|spadPrompt|))
```

————-

Chapter 44

)savesystem help page Command

44.1 savesystem help page man page

— savesystem.help —

```
=======================================================================
A.8.  )savesystem
=======================================================================

User Level Required:  interpreter


Command Syntax:

  - )savesystem filename

Command Description:

 This command is used to save an AXIOM image to disk.  This creates an
executable file which, when started, has everything loaded into it
that was there when the image was saved.  Thus, after executing commands
which cause the loading of some packages, the command:

)savesystem /tmp/savesys

will create an image that can be restarted  with the UNIX command:

axiom -ws /tmp/savesys
```

```
This new system will not need to reload the packages and domains that
were already loaded when the system was saved.

There is currently a restriction that only systems started with the
command "AXIOMsys" may be saved.
```

———-

44.1.1 defun The)savesystem command

[helpSpad2Cmd p568]
[spad-save p979]

— defun savesystem —

```
(defun |savesystem| (arg)
  (if (or (not (eql (|#| arg) 1)) (null (symbolp (car arg))))
    (|helpSpad2Cmd| '(|savesystem|))
    (spad-save (symbol-name (car arg)))))
```

———-

Chapter 45

)set help page Command

45.1 set help page man page

— set.help —

```
====================================================================
A.21.  )set
====================================================================

User Level Required:  interpreter

Command Syntax:

  -  )set
  -  )set label1 [... labelN]
  -  )set label1 [... labelN] newValue

Command Description:

The )set command is used to view or set system variables that control what
messages are displayed, the type of output desired, the status of the history
facility, the way AXIOM user functions are cached, and so on. Since this
collection is very large, we will not discuss them here. Rather, we will show
how the facility is used. We urge you to explore the )set options to
familiarize yourself with how you can modify your AXIOM working environment.
There is a HyperDoc version of this same facility available from the main
HyperDoc menu. Click [here] to go to it.

The )set command is command-driven with a menu display. It is
tree-structured. To see all top-level nodes, issue )set by itself.

)set
```

```
Variables with values have them displayed near the right margin. Subtrees of
selections have ‘‘...’’ displayed in the value field. For example, there are
many kinds of messages, so issue )set message to see the choices.

)set message

The current setting for the variable that displays whether computation times
are displayed is visible in the menu displayed by the last command. To see
more information, issue

)set message time

This shows that time printing is on now. To turn it off, issue

)set message time off

As noted above, not all settings have so many qualifiers. For example, to
change the )quit command to being unprotected (that is, you will not be
prompted for verification), you need only issue

)set quit unprotected

Also See:
o )quit
```

[1]

45.2 Overview

This section contains tree of information used to initialize the **)set** command in the interpreter. The current list is:

```
Variable     Description                              Current Value
-------------------------------------------------------------------
compile    Library compiler options                      ...
breakmode  execute break processing on error           break
expose     control interpreter constructor exposure    ...
functions  some interpreter function options           ...
fortran    view and set options for FORTRAN output     ...
kernel     library functions built into the kernel for
           efficiency                                  ...
hyperdoc   options in using HyperDoc                   ...
```

[1] “quit” (42.2.1 p 634)

```
help       view and set some help options             ...
history    save workspace values in a history file    on
messages   show messages for various system features  ...
naglink    options for NAGLink                        ...
output     view and set some output options           ...
quit       protected or unprotected quit              unprotected
streams    set some options for working with streams  ...
system     set some system development variables      ...
userlevel  operation access level of system user      development

Variables with current values of ... have further sub-options.
For example, issue )set  system to see what the options are
for system. For more information, issue )help set .
```

45.3 Variables Used

45.4 Functions

45.4.1 Initialize the set variables

The argument settree is initially the `$setOption` variable. The fourth element is a union-style switch symbol. The fifth element is usually a variable to set. The sixth element is a subtree to recurse for the TREE switch. The seventh element is usually the default value. For more detailed explanations see the list structure section 45.5. [sayMSG p329]
[literals p**??**]
[translateYesNo2TrueFalse p650]
[tree p**??**]
[initializeSetVariables p645]

— defun initializeSetVariables —

```
(defun |initializeSetVariables| (settree)
 "Initialize the set variables"
 (dolist (setdata settree)
  (case (fourth setdata)
   (function
      (if (functionp (fifth setdata))
        (funcall (fifth setdata) '|%initialize%|)
        (|sayMSG| (concatenate 'string "   Function not implemented. "
         (package-name *package*) ":" (string (fifth setdata))))))
   (integer (set (fifth setdata) (seventh setdata)))
   (string  (set (fifth setdata) (seventh setdata)))
   (literals
       (set (fifth setdata) (|translateYesNo2TrueFalse| (seventh setdata))))
   (tree  (|initializeSetVariables| (sixth setdata))))))
```

———-

45.4.2 Reset the workspace variables

[copy p??]
[initializeSetVariables p645]
[/countlist p??]
[/editfile p511]
[/sourcefiles p??]
[/pretty p??]
[/spacelist p??]
[/timerlist p??]
[$sourceFiles p??]
[$existingFiles p??]
[$functionTable p498]
[$boot p25]
[$compileMapFlag p??]
[$echoLineStack p??]
[$operationNameList p??]
[$slamFlag p??]
[$CommandSynonymAlist p474]
[$InitialCommandSynonymAlist p472]
[$UserAbbreviationsAlist p??]
[$msgAlist p324]
[$msgDatabase p??]
[$msgDatabaseName p324]
[$dependeeClosureAlist p??]
[$IOindex p??]
[$coerceIntByMapCounter p??]
[$e p??]
[$env p??]
[$setOptions p??]

— **defun resetWorkspaceVariables** —

```
(defun |resetWorkspaceVariables| ()
 "Reset the workspace variables"
 (declare (special /countlist /editfile /sourcefiles |$sourceFiles| /pretty
    /spacelist /timerlist |$existingFiles| |$functionTable| $boot
    |$compileMapFlag| |$echoLineStack| |$operationNameList| |$slamFlag|
    |$CommandSynonymAlist| |$InitialCommandSynonymAlist|
    |$UserAbbreviationsAlist| |$msgAlist| |$msgDatabase| |$msgDatabaseName|
    |$dependeeClosureAlist| |$IOindex| |$coerceIntByMapCounter| |$e| |$env|
    |$setOptions|))
```

```
(setq /countlist nil)
(setq /editfile nil)
(setq /sourcefiles nil)
(setq |$sourceFiles| nil)
(setq /pretty nil)
(setq /spacelist nil)
(setq /timerlist nil)
(setq |$existingFiles| (make-hash-table :test #'equal))
(setq |$functionTable| nil)
(setq $boot nil)
(setq |$compileMapFlag| nil)
(setq |$echoLineStack| nil)
(setq |$operationNameList| nil)
(setq |$slamFlag| nil)
(setq |$CommandSynonymAlist| (copy |$InitialCommandSynonymAlist|))
(setq |$UserAbbreviationsAlist| nil)
(setq |$msgAlist| nil)
(setq |$msgDatabase| nil)
(setq |$msgDatabaseName| nil)
(setq |$dependeeClosureAlist| nil)
(setq |$IOindex| 1)
(setq |$coerceIntByMapCounter| 0)
(setq |$e| (cons (cons nil nil) nil))
(setq |$env| (cons (cons nil nil) nil))
(|initializeSetVariables| |$setOptions|))
```

———-

45.4.3 Display the set option information

[displaySetVariableSettings p649]
[centerAndHighlight p**??**]
[concat p1041]
[object2String p**??**]
[specialChar p970]
[sayBrightly p**??**]
[bright p**??**]
[sayMSG p329]
[boot-equal p**??**]
[sayMessage p**??**]
[eval p**??**]
[literals p**??**]
[translateTrueFalse2YesNo p651]
[$linelength p765]

— **defun displaySetOptionInformation** —

```
(defun |displaySetOptionInformation| (arg setdata)
 "Display the set option information"
 (let (current)
 (declare (special $linelength))
  (cond
   ((eq (fourth setdata) 'tree)
    (|displaySetVariableSettings| (sixth setdata) (first setdata)))
   (t
    (|centerAndHighlight|
     (concat "The " (|object2String| arg) " Option")
      $linelength (|specialChar| '|hbar|))
    (|sayBrightly|
     `(|%l| ,@(|bright| "Description:") ,(second setdata)))
    (case (fourth setdata)
     (function
      (terpri)
      (if (functionp (fifth setdata))
        (funcall (fifth setdata) '|%describe%|)
        (|sayMSG| "   Function not implemented.")))
     (integer
      (|sayMessage|
        `(" The" ,@(|bright| arg) "option"
          " may be followed by an integer in the range"
          ,@(|bright| (elt (sixth setdata) 0)) "to"
          |%l| ,@(|bright| (elt (sixth setdata) 1)) "inclusive."
          " The current setting is" ,@(|bright| (|eval| (fifth setdata))))))
     (string
      (|sayMessage|
        `(" The" ,@(|bright| arg) "option"
          " is followed by a string enclosed in double quote marks."
          '|%l| " The current setting is"
          ,@(|bright| (list '|"| (|eval| (fifth setdata)) '|"|)))))
     (literals
       (|sayMessage|
        `(" The" ,@(|bright| arg) "option"
              " may be followed by any one of the following:"))
       (setq current
        (|translateTrueFalse2YesNo| (|eval| (fifth setdata))))
       (dolist (name (sixth setdata))
         (if (boot-equal name current)
           (|sayBrightly| `( " ->" ,@(|bright| (|object2String| name))))
           (|sayBrightly| (list "    " (|object2String| name)))))
       (|sayMessage| " The current setting is indicated."))))))
```

———-

45.4.4 Display the set variable settings

[concat p1041]
[object2String p??]
[centerAndHighlight p??]
[sayBrightly p??]
[say p??]
[fillerSpaces p20]
[specialChar p970]
[concat p1041]
[satisfiesUserLevel p447]
[spaddifference p??]
[poundsign p??]
[eval p??]
[bright p??]
[literals p??]
[translateTrueFalse2YesNo p651]
[tree p??]
[$linelength p765]

— **defun displaySetVariableSettings** —

```
(defun |displaySetVariableSettings| (settree label)
 "Display the set variable settings"
 (let (setoption opt subtree subname)
 (declare (special $linelength))
  (if (eq label '||)
    (setq label ")set")
    (setq label (concat "  " (|object2String| label) " ")))
  (|centerAndHighlight|
    (concat "Current Values of" label " Variables") $linelength '| |)
  (terpri)
  (|sayBrightly|
   (list "Variable     " "Description                                  "
         "Current Value" ))
  (say (|fillerSpaces| $linelength (|specialChar| '|hbar|)))
  (setq subtree nil)
  (dolist (setdata settree)
   (when (|satisfiesUserLevel| (third setdata))
     (setq setoption (|object2String| (first setdata)))
     (setq setoption
      (concat setoption
       (|fillerSpaces| (spaddifference 13 (|#| setoption)) " ")
       (second setdata)))
     (setq setoption
      (concat setoption
       (|fillerSpaces| (spaddifference 55 (|#| setoption)) " ")))
     (case (fourth setdata)
```

```
      (function
       (setq opt
        (if (functionp (fifth setdata))
         (funcall (fifth setdata) '|%display%|)
         "unimplemented"))
       (cond
        ((consp opt)
         (setq opt
          (do ((t2 opt (cdr t2)) t1 (o nil))
              ((or (atom t2) (progn (setq o (car t2)) nil)) t1)
           (setq t1 (append t1 (cons o (cons " " nil)))))))))
       (|sayBrightly| (|concat| setoption '|%b| opt '|%d|)))
      (string
       (setq opt (|object2String| (|eval| (fifth setdata))))
       (|sayBrightly| `(,setoption ,@(|bright| opt))))
      (integer
       (setq opt (|object2String| (|eval| (fifth setdata))))
       (|sayBrightly| `(,setoption ,@(|bright| opt))))
      (literals
       (setq opt (|object2String|
            (|translateTrueFalse2YesNo| (|eval| (fifth setdata)))))
       (|sayBrightly| `(,setoption ,@(|bright| opt))))
      (TREE
         (|sayBrightly| `(,setoption ,@(|bright| "...")))
         (setq subtree t)
         (setq subname (|object2String| (first setdata)))))))
  (terpri)
  (when subtree
     (|sayBrightly|
      `("Variables with current values of" ,@(|bright| "...")
            "have further sub-options. For example,"))
     (|sayBrightly|
      `("issue" ,@(|bright| ")set ") ,subname
        " to see what the options are for" ,@(|bright| subname) "."
        |%l| "For more information, issue" ,@(|bright| ")help set") ".")))))
```

———-

45.4.5 Translate options values to t or nil

[member p1042]

— defun translateYesNo2TrueFalse —

```
(defun |translateYesNo2TrueFalse| (x)
 "Translate options values to t or nil"
 (cond
  ((|member| x '(|yes| |on|)) t)
```

```
  ((|member| x '(|no| |off|)) nil)
  (t x)))
```

———-

45.4.6 Translate t or nil to option values

— **defun translateTrueFalse2YesNo** —

```
(defun |translateTrueFalse2YesNo| (x)
 "Translate t or nil to option values"
 (cond
  ((eq x t) '|on|)
  ((null x) '|off|)
  (t x)))
```

———-

45.5 The list structure

The structure of each list item consists of 7 items. Consider this example:

```
  (userlevel
   "operation access level of system user"
   interpreter
   LITERALS
   $UserLevel
   (interpreter compiler development)
   development)
```

The list looks like (the names in bold are accessor names that can be found in **property.lisp.pamphlet**[**1**]. Look for "setName".):

1 *Name* the keyword the user will see. In this example the user would say "**)set output userlevel**".

2 *Label* the message the user will see. In this example the user would see "operation access level of system user".

3 *Level* the level where the command will be accepted. There are three levels: interpreter, compiler, development. These commands are restricted to keep the user from causing damage.

4 *Type* a symbol, one of **FUNCTION**, **INTEGER**, **STRING**, **LITERALS**, **FILENAME** or **TREE**.

5 *Var*

FUNCTION is the function to call

INTEGER is the variable holding the current user setting.

STRING is the variable holding the current user setting.

LITERALS variable which holds the current user setting.

FILENAME is the variable that holds the current user setting.

TREE

6 *Leaf*

FUNCTION is the list of all possible values

INTEGER is the range of possible values

STRING is a list of all possible values

LITERALS is a list of all of the possible values

FILENAME is the function to check the filename

TREE

7 *Def* is the default value

FUNCTION is the default setting

INTEGER is the default setting

STRING is the default setting

LITERALS is the default setting

FILENAME is the default value

TREE

45.6 breakmode

```
-------------------- The breakmode Option ---------------------

 Description: execute break processing on error

 The breakmode option may be followed by any one of the
 following:

    nobreak
 -> break
    query
    resume
```

```
    fastlinks
    quit

 The current setting is indicated.
```

45.6.1 defvar $BreakMode

— **initvars** —

```
(defvar |$BreakMode| '|nobreak| "execute break processing on error")
```

————-

— **breakmode** —

```
  (|breakmode|
   "execute break processing on error"
   |interpreter|
   LITERALS
   |$BreakMode|
   (|nobreak| |break| |query| |resume| |fastlinks| |quit|)
   |nobreak|)         ; needed to avoid possible startup looping
```

————-

45.7 debug

```
                Current Values of debug Variables

Variable     Description                           Current Value
-----------------------------------------------------------------
lambdatype   Show type information for #1 syntax     off
dalymode     Interpret leading open paren as lisp    off
```

— **debug** —

```
   (|debug|
    "debug options"
    |interpreter|
    TREE
    |novar|
```

```
    (
\getchunk{debuglambdatype}
\getchunk{debugdalymode}
    ))
```

———-

45.8 debug lambda type

```
---------------------- The lambdatype Option ----------------------

 Description: Show type information for #1 syntax
```

45.8.1 defvar $lambdatype

— **initvars** —

```
(defvar $lambdatype nil "show type information for #1 syntax")
```

———-

— **debuglambdatype** —

```
      (|lambdatype|
       "show type information for #1 syntax"
       |interpreter|
       LITERALS
       $lambdatype
       (|on| |off|)
       |off|)
```

———-

45.9 debug dalymode

The `$dalymode` variable is used in a case statement in intloopReadConsole. This variable can be set to any non-nil value. When not nil the interpreter will send any line that begins with an "(" to be sent to the underlying lisp. This is useful for debugging Axiom. The normal value of this variable is NIL.

This variable was created as an alternative to prefixing every lisp command with)lisp. When doing a lot of debugging this is tedious and error prone. This variable was created to shortcut

that process. Clearly it breaks some semantics of the language accepted by the interpreter as parens are used for grouping expressions.

```
---------------------- The dalymode Option ----------------------

 Description: Interpret leading open paren as lisp
```

45.9.1 defvar $dalymode

— **initvars** —

```
(defvar $dalymode nil "Interpret leading open paren as lisp")
```

———-

— **debugdalymode** —

```
      (|dalymode|
       "Interpret leading open paren as lisp"
       |interpreter|
       LITERALS
       $dalymode
       (|on| |off|)
       |off|)
```

———-

45.10 compile

Current Values of compiler Variables

Variable	Description	Current Value
output	library in which to place compiled code	
input	controls libraries from which to load compiled code	

— **compile** —

```
   (|compiler|
    "Library compiler options"
```

```
    |interpreter|
    TREE
    |novar|
    (
\getchunk{compileoutput}
\getchunk{compileinput}
    ))
```

———-

45.11 compile output

```
---------------------- The output Option ----------------------

 Description: library in which to place compiled code
```

— **compileoutput** —

```
      (|output|
       "library in which to place compiled code"
       |interpreter|
       FUNCTION
       |setOutputLibrary|
       NIL
       |htSetOutputLibrary|
       )
```

———-

45.12 Variables Used

45.13 Functions

45.13.1 The set output command handler

[poundsign p**??**]
[describeOutputLibraryArgs p657]
[filep p**??**]
[openOutputLibrary p658]
[$outputLibraryName p**??**]

— **defun setOutputLibrary** —

```
(defun |setOutputLibrary| (arg)
 "The set output command handler"
 (let (fn)
 (declare (special |$outputLibraryName|))
   (cond
    ((eq arg '|%initialize%|) (setq |$outputLibraryName| nil))
    ((eq arg '|%display%|) (or |$outputLibraryName| "user.lib"))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?) (/= (|#| arg) 1))
     (|describeOutputLibraryArgs|))
    (t
     (when (probe-file (setq fn (princ-to-string (car arg))))
       (setq fn (truename fn)))
     (|openOutputLibrary| (setq |$outputLibraryName| fn))))))
```

———-

45.13.2 Describe the set output library arguments

[sayBrightly p??]

— **defun describeOutputLibraryArgs** —

```
(defun |describeOutputLibraryArgs| ()
 "Describe the set output library arguments"
 (|sayBrightly| (list
  '|%b| ")set compile output library"
  '|%d| "is used to tell the compiler where to place"
  '|%l| "compiled code generated by the library compiler.  By default it goes"
  '|%l| "in a file called"
  '|%b| "user.lib"
  '|%d| "in the current directory.")))
```

———-

45.13.3 defvar $output-library

— **initvars** —

```
(defvar output-library nil)
```

———-

45.13.4 Open the output library

The input-libraries and output-library are now truename based. [dropInputLibrary p661]
[output-library p657]
[input-libraries p660]

— **defun openOutputLibrary** —

```
(defun |openOutputLibrary| (lib)
 "Open the output library"
 (declare (special output-library input-libraries))
  (|dropInputLibrary| lib)
  (setq output-library (truename lib))
  (push output-library input-libraries))
```

———-

45.14 compile input

```
---------------------- The input Option -----------------------

 Description: controls libraries from which to load compiled code

 )set compile input add library is used to tell AXIOM to add
  library to the front of the path which determines where
  compiled code is loaded from.
 )set compile input drop library is used to tell AXIOM to remove
  library from this path.
```

— **compileinput** —

```
     (|input|
      "controls libraries from which to load compiled code"
      |interpreter|
      FUNCTION
      |setInputLibrary|
      NIL
      |htSetInputLibrary|)
```

———-

45.15 Variables Used

45.16 Functions

45.16.1 The set input library command handler

The input-libraries is now maintained as a list of truenames. [describeInputLibraryArgs p660]
[qcar p??]
[qcdr p??]
[selectOptionLC p475]
[addInputLibrary p660]
[dropInputLibrary p661]
[setInputLibrary p659]
[input-libraries p660]

— **defun setInputLibrary** —

```
(defun |setInputLibrary| (arg)
 "The set input library command handler"
 (declare (special input-libraries))
 (let (tmp1 filename act)
  (cond
   ((eq arg '|%initialize%|) t)
   ((eq arg '|%display%|) (mapcar #'namestring input-libraries))
   ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
    (|describeInputLibraryArgs|))
   ((and (consp arg)
         (progn
          (setq act (qcar arg))
          (setq tmp1 (qcdr arg))
          (and (consp tmp1)
               (eq (qcdr tmp1) nil)
               (progn (setq filename (qcar tmp1)) t)))
         (setq act (|selectOptionLC| act '(|add| |drop|) nil)))
    (cond
     ((eq act '|add|)
      (|addInputLibrary| (truename (princ-to-string filename))))
     ((eq act '|drop|)
      (|dropInputLibrary| (truename (princ-to-string filename))))))
   (t (|setInputLibrary| nil)))))
```

————-

45.16.2 Describe the set input library arguments

[sayBrightly p??]

— **defun describeInputLibraryArgs** —

```
(defun |describeInputLibraryArgs| ()
 "Describe the set input library arguments"
 (|sayBrightly| (list
  '|%b| ")set compile input add library"
  '|%d| "is used to tell AXIOM to add"
  '|%b| "library"
  '|%d| "to"
  '|%l| "the front of the path used to find compile code."
  '|%l|
  '|%b| ")set compile input drop library"
  '|%d| "is used to tell AXIOM to remove"
  '|%b| "library"
  '|%d|
  '|%l| "from this path.")))
```

————-

45.16.3 Add the input library to the list

The input-libraries variable is now maintained as a list of truenames. [dropInputLibrary p661]
[input-libraries p660]

— **defun addInputLibrary** —

```
(defun |addInputLibrary| (lib)
 "Add the input library to the list"
 (declare (special input-libraries))
  (|dropInputLibrary| lib)
  (push (truename lib) input-libraries))
```

————-

45.16.4 defvar $input-libraries

— **initvars** —

```
(defvar input-libraries nil)
```

———-

45.16.5 Drop an input library from the list

[input-libraries p660]

— **defun dropInputLibrary** —

```
(defun |dropInputLibrary| (lib)
 "Drop an input library from the list"
 (declare (special input-libraries))
 (setq input-libraries (delete (truename lib) input-libraries :test #'equal)))
```

———-

45.17 expose

```
---------------------- The expose Option ----------------------

 Description: control interpreter constructor exposure

   The following groups are explicitly exposed in the current
   frame (called initial ):
                                    basic
                                categories
                                  naglink
                                   anna

   The following constructors are explicitly exposed in the
   current frame:
               there are no explicitly exposed constructors

   The following constructors are explicitly hidden in the
   current frame:
                there are no explicitly hidden constructors

   When )set expose is followed by no arguments, the information
   you now see is displayed. When followed by the initialize
   argument, the exposure group data in the file interp.exposed
   is read and is then available. The arguments add and drop are
   used to add or drop exposure groups or explicit constructors
   from the local frame exposure data. Issue
                  )set expose add    or    )set expose drop
   for more information.
```

— **expose** —

```
(|expose|
 "control interpreter constructor exposure"
 |interpreter|
 FUNCTION
 |setExpose|
 NIL
 |htSetExpose|)
```

———-

45.18 Variables Used

NOTE: If you add new algebra you must also update this list otherwise the new algebra won't be loaded by the interpreter when needed.

45.18.1 defvar $globalExposureGroupAlist

— **initvars** —

```
(defvar |$globalExposureGroupAlist|
 '(
 ;;define the groups |basic| |naglink| |anna| |categories| |Hidden| |defaults|
  (|basic|
   (|AffineAlgebraicSetComputeWithGroebnerBasis| . AFALGGRO)
   (|AffineAlgebraicSetComputeWithResultant| . AFALGRES)
   (|AffinePlane| . AFFPL)
   (|AffinePlaneOverPseudoAlgebraicClosureOfFiniteField| . AFFPLPS)
   (|AffineSpace| . AFFSP)
   (|AlgebraicManipulations| . ALGMANIP)
   (|AlgebraicNumber| . AN)
   (|AlgFactor| . ALGFACT)
   (|AlgebraicMultFact| . ALGMFACT)
   (|AlgebraPackage| . ALGPKG)
   (|AlgebraGivenByStructuralConstants| . ALGSC)
   (|Any| . ANY)
   (|AnyFunctions1| . ANY1)
   (|ApplicationProgramInterface| . API)
   (|ArrayStack| . ASTACK)
   (|AssociatedJordanAlgebra| . JORDAN)
   (|AssociatedLieAlgebra| . LIE)
   (|AttachPredicates| . PMPRED)
   (|AxiomServer| . AXSERV)
   (|BalancedBinaryTree| . BBTREE)
```

```
(|BasicStochasticDifferential| . BSD)
(|BasicOperator| . BOP)
(|BasicOperatorFunctions1| . BOP1)
(|Bezier| . BEZIER)
(|BinaryExpansion| . BINARY)
(|BinaryFile| . BINFILE)
(|BinarySearchTree| . BSTREE)
(|BinaryTournament| . BTOURN)
(|BinaryTree| . BTREE)
(|Bits| . BITS)
(|BlasLevelOne| . BLAS1)
(|BlowUpPackage| . BLUPPACK)
(|BlowUpWithHamburgerNoether| . BLHN)
(|BlowUpWithQuadTrans| . BLQT)
(|Boolean| . BOOLEAN)
(|CardinalNumber| . CARD)
(|CartesianTensor| . CARTEN)
(|CartesianTensorFunctions2| . CARTEN2)
(|Character| . CHAR)
(|CharacterClass| . CCLASS)
(|CharacteristicPolynomialPackage| . CHARPOL)
(|CliffordAlgebra| . CLIF)
(|Color| . COLOR)
(|CommonDenominator| . CDEN)
(|Commutator| . COMM)
(|Complex| . COMPLEX)
(|ComplexDoubleFloatMatrix| . CDFMAT)
(|ComplexDoubleFloatVector| . CDFVEC)
(|ComplexFactorization| . COMPFACT)
(|ComplexFunctions2| . COMPLEX2)
(|ComplexRootPackage| . CMPLXRT)
(|ComplexTrigonometricManipulations| . CTRIGMNP)
(|ContinuedFraction| . CONTFRAC)
(|CoordinateSystems| . COORDSYS)
(|CRApackage| . CRAPACK)
(|CycleIndicators| . CYCLES)
(|Database| . DBASE)
(|DataList| . DLIST)
(|DecimalExpansion| . DECIMAL)
(|DenavitHartenbergMatrix| . DHMATRIX)
(|Dequeue| . DEQUEUE)
(|DesingTree| . DSTREE)
(|DesingTreePackage| . DTP)
(|DiophantineSolutionPackage| . DIOSP)
(|DirichletRing| . DIRRING)
(|DirectProductFunctions2| . DIRPROD2)
(|DisplayPackage| . DISPLAY)
(|DistinctDegreeFactorize| . DDFACT)
(|Divisor| . DIV)
(|DoubleFloat| . DFLOAT)
```

```
 (|DoubleFloatMatrix| . DFMAT)
 (|DoubleFloatVector| . DFVEC)
 (|DoubleFloatSpecialFunctions| . DFSFUN)
 (|DrawComplex| . DRAWCX)
 (|DrawNumericHack| . DRAWHACK)
 (|DrawOption| . DROPT)
 (|EigenPackage| . EP)
 (|ElementaryFunctionDefiniteIntegration| . DEFINTEF)
 (|ElementaryFunctionLODESolver| . LODEEF)
 (|ElementaryFunctionODESolver| . ODEEF)
 (|ElementaryFunctionSign| . SIGNEF)
 (|ElementaryFunctionStructurePackage| . EFSTRUC)
 (|Equation| . EQ)
 (|EquationFunctions2| . EQ2)
 (|ErrorFunctions| . ERROR)
 (|EuclideanGroebnerBasisPackage| . GBEUCLID)
 (|Exit| . EXIT)
 (|Export3D| . EXP3D)
 (|Expression| . EXPR)
 (|ExpressionFunctions2| . EXPR2)
 (|ExpressionSolve| . EXPRSOL)
 (|ExpressionSpaceFunctions2| . ES2)
 (|ExpressionSpaceODESolver| . EXPRODE)
 (|ExpressionToOpenMath| . OMEXPR)
 (|ExpressionToUnivariatePowerSeries| . EXPR2UPS)
 (|Factored| . FR)
 (|FactoredFunctions2| . FR2)
(|FactorisationOverPseudoAlgebraicClosureOfAlgExtOfRationalNumber| . FACTEXT)
 (|FactorisationOverPseudoAlgebraicClosureOfRationalNumber| . FACTRN)
 (|File| . FILE)
 (|FileName| . FNAME)
 (|FiniteAbelianMonoidRingFunctions2| . FAMR2)
 (|FiniteDivisorFunctions2| . FDIV2)
 (|FiniteFieldFactorizationWithSizeParseBySideEffect| . FFFACTSE)
 (|FiniteField| . FF)
 (|FiniteFieldCyclicGroup| . FFCG)
 (|FiniteFieldPolynomialPackage2| . FFPOLY2)
 (|FiniteFieldNormalBasis| . FFNB)
 (|FiniteFieldHomomorphisms| . FFHOM)
 (|FiniteFieldSquareFreeDecomposition| . FFSQFR)
 (|FiniteLinearAggregateFunctions2| . FLAGG2)
 (|FiniteLinearAggregateSort| . FLASORT)
 (|FiniteSetAggregateFunctions2| . FSAGG2)
 (|FlexibleArray| . FARRAY)
 (|Float| . FLOAT)
 (|FloatingRealPackage| . FLOATRP)
 (|FloatingComplexPackage| . FLOATCP)
 (|FourierSeries| . FSERIES)
 (|Fraction| . FRAC)
 (|FractionalIdealFunctions2| . FRIDEAL2)
```

```
(|FractionFreeFastGaussian| . FFFG)
(|FractionFreeFastGaussianFractions| . FFFGF)
(|FractionFunctions2| . FRAC2)
(|FreeNilpotentLie| . FNLA)
(|FullPartialFractionExpansion| . FPARFRAC)
(|FunctionFieldCategoryFunctions2| . FFCAT2)
(|FunctionSpaceAssertions| . PMASSFS)
(|FunctionSpaceAttachPredicates| . PMPREDFS)
(|FunctionSpaceComplexIntegration| . FSCINT)
(|FunctionSpaceFunctions2| . FS2)
(|FunctionSpaceIntegration| . FSINT)
(|FunctionSpacePrimitiveElement| . FSPRMELT)
(|FunctionSpaceSum| . SUMFS)
(|GaussianFactorizationPackage| . GAUSSFAC)
(|GeneralPackageForAlgebraicFunctionField| . GPAFF)
(|GeneralUnivariatePowerSeries| . GSERIES)
(|GenerateUnivariatePowerSeries| . GENUPS)
(|GnuDraw| . GDRAW)
(|GraphicsDefaults| . GRDEF)
(|GroebnerPackage| . GB)
(|GroebnerFactorizationPackage| . GBF)
(|Guess| . GUESS)
(|GuessAlgebraicNumber| . GUESSAN)
(|GuessFinite| . GUESSF)
(|GuessFiniteFunctions| . GUESSF1)
(|GuessInteger| . GUESSINT)
(|GuessOption| . GOPT)
(|GuessPolynomial| . GUESSP)
(|GuessUnivariatePolynomial| . GUESSUP)
(|HallBasis| . HB)
(|Heap| . HEAP)
(|HexadecimalExpansion| . HEXADEC)
(|HTMLFormat| . HTMLFORM)
(|IdealDecompositionPackage| . IDECOMP)
(|IndexCard| . ICARD)
(|InfClsPt| . ICP)
(|InfiniteProductCharacteristicZero| . INFPROD0)
(|InfiniteProductFiniteField| . INPRODFF)
(|InfiniteProductPrimeField| . INPRODPF)
(|InfiniteTuple| . ITUPLE)
(|InfiniteTupleFunctions2| . ITFUN2)
(|InfiniteTupleFunctions3| . ITFUN3)
(|InfinitlyClosePoint| . INFCLSPT)
(|InfinitlyClosePointOverPseudoAlgebraicClosureOfFiniteField| . INFCLSPS)
(|Infinity| . INFINITY)
(|Integer| . INT)
(|IntegerCombinatoricFunctions| . COMBINAT)
(|IntegerLinearDependence| . ZLINDEP)
(|IntegerNumberTheoryFunctions| . INTHEORY)
(|IntegerPrimesPackage| . PRIMES)
```

```
(|IntegerRetractions| . INTRET)
(|IntegerRoots| . IROOT)
(|IntegrationResultFunctions2| . IR2)
(|IntegrationResultRFToFunction| . IRRF2F)
(|IntegrationResultToFunction| . IR2F)
(|InterfaceGroebnerPackage| . INTERGB)
(|InterpolateFormsPackage| . INTFRSP)
(|IntersectionDivisorPackage| . INTDIVP)
(|Interval| . INTRVL)
(|InventorDataSink| . IVDATA)
(|InventorViewPort| . IVVIEW)
(|InventorRenderPackage| . IVREND)
(|InverseLaplaceTransform| . INVLAPLA)
(|IrrRepSymNatPackage| . IRSN)
(|KernelFunctions2| . KERNEL2)
(|KeyedAccessFile| . KAFILE)
(|LaplaceTransform| . LAPLACE)
(|LazardMorenoSolvingPackage| . LAZM3PK)
(|Library| . LIB)
(|LieSquareMatrix| . LSQM)
(|LinearOrdinaryDifferentialOperator| . LODO)
(|LinearSystemMatrixPackage| . LSMP)
(|LinearSystemMatrixPackage1| . LSMP1)
(|LinearSystemFromPowerSeriesPackage| . LISYSER)
(|LinearSystemPolynomialPackage| . LSPP)
(|List| . LIST)
(|LinesOpPack| . LOP)
(|ListFunctions2| . LIST2)
(|ListFunctions3| . LIST3)
(|ListToMap| . LIST2MAP)
(|LocalParametrizationOfSimplePointPackage| . LPARSPT)
(|MakeFloatCompiledFunction| . MKFLCFN)
(|MakeFunction| . MKFUNC)
(|MakeRecord| . MKRECORD)
(|MappingPackage1| . MAPPKG1)
(|MappingPackage2| . MAPPKG2)
(|MappingPackage3| . MAPPKG3)
(|MappingPackage4| . MAPPKG4)
(|MathMLFormat| . MMLFORM)
(|Matrix| . MATRIX)
(|MatrixCategoryFunctions2| . MATCAT2)
(|MatrixCommonDenominator| . MCDEN)
(|MatrixLinearAlgebraFunctions| . MATLIN)
(|MergeThing| . MTHING)
(|ModularDistinctDegreeFactorizer| . MDDFACT)
(|ModuleOperator| . MODOP)
(|MonoidRingFunctions2| . MRF2)
(|MoreSystemCommands| . MSYSCMD)
(|MPolyCatFunctions2| . MPC2)
(|MPolyCatRationalFunctionFactorizer| . MPRFF)
```

```
(|Multiset| . MSET)
(|MultivariateFactorize| . MULTFACT)
(|MultivariatePolynomial| . MPOLY)
(|MultFiniteFactorize| . MFINFACT)
(|MyUnivariatePolynomial| . MYUP)
(|MyExpression| . MYEXPR)
(|NeitherSparseOrDensePowerSeries| . NSDPS)
(|NewtonPolygon| . NPOLYGON)
(|NoneFunctions1| . NONE1)
(|NonNegativeInteger| . NNI)
(|NottinghamGroup| . NOTTING)
(|NormalizationPackage| . NORMPK)
(|NormInMonogenicAlgebra| . NORMMA)
(|NumberTheoreticPolynomialFunctions| . NTPOLFN)
(|Numeric| . NUMERIC)
(|NumericalOrdinaryDifferentialEquations| . NUMODE)
(|NumericalQuadrature| . NUMQUAD)
(|NumericComplexEigenPackage| . NCEP)
(|NumericRealEigenPackage| . NREP)
(|NumericContinuedFraction| . NCNTFRAC)
(|Octonion| . OCT)
(|OctonionCategoryFunctions2| . OCTCT2)
(|OneDimensionalArray| . ARRAY1)
(|OneDimensionalArrayFunctions2| . ARRAY12)
(|OnePointCompletion| . ONECOMP)
(|OnePointCompletionFunctions2| . ONECOMP2)
(|OpenMathConnection| . OMCONN)
(|OpenMathDevice| . OMDEV)
(|OpenMathEncoding| . OMENC)
(|OpenMathError| . OMERR)
(|OpenMathErrorKind| . OMERRK)
(|OpenMathPackage| . OMPKG)
(|OpenMathServerPackage| . OMSERVER)
(|OperationsQuery| . OPQUERY)
(|OrderedCompletion| . ORDCOMP)
(|OrderedCompletionFunctions2| . ORDCOMP2)
(|OrdinaryDifferentialRing| . ODR)
(|OrdSetInts| . OSI)
(|OrthogonalPolynomialFunctions| . ORTHPOL)
(|OutputPackage| . OUT)
(|PackageForAlgebraicFunctionField| . PAFF)
(|PackageForAlgebraicFunctionFieldOverFiniteField| . PAFFFF)
(|PackageForPoly| . PFORP)
(|PadeApproximantPackage| . PADEPAC)
(|Palette| . PALETTE)
(|PartialFraction| . PFR)
(|PatternFunctions2| . PATTERN2)
(|ParametricPlaneCurve| . PARPCURV)
(|ParametricSpaceCurve| . PARSCURV)
(|ParametricSurface| . PARSURF)
```

```
(|ParametricPlaneCurveFunctions2| . PARPC2)
(|ParametricSpaceCurveFunctions2| . PARSC2)
(|ParametricSurfaceFunctions2| . PARSU2)
(|ParametrizationPackage| . PARAMP)
(|PartitionsAndPermutations| . PARTPERM)
(|PatternMatch| . PATMATCH)
(|PatternMatchAssertions| . PMASS)
(|PatternMatchResultFunctions2| . PATRES2)
(|PendantTree| . PENDTREE)
(|Permanent| . PERMAN)
(|PermutationGroupExamples| . PGE)
(|PermutationGroup| . PERMGRP)
(|Permutation| . PERM)
(|Pi| . HACKPI)
(|PiCoercions| . PICOERCE)
(|Places| . PLACES)
(|PlacesOverPseudoAlgebraicClosureOfFiniteField| . PLACESPS)
(|Plcs| . PLCS)
(|PointFunctions2| . PTFUNC2)
(|PolyGroebner| . PGROEB)
(|Polynomial| . POLY)
(|PolynomialAN2Expression| . PAN2EXPR)
(|PolynomialComposition| . PCOMP)
(|PolynomialDecomposition| . PDECOMP)
(|PolynomialFunctions2| . POLY2)
(|PolynomialIdeals| . IDEAL)
(|PolynomialPackageForCurve| . PLPKCRV)
(|PolynomialToUnivariatePolynomial| . POLY2UP)
(|PositiveInteger| . PI)
(|PowerSeriesLimitPackage| . LIMITPS)
(|PrimeField| . PF)
(|PrimitiveArrayFunctions2| . PRIMARR2)
(|PrintPackage| . PRINT)
(|ProjectiveAlgebraicSetPackage| . PRJALGPK)
(|ProjectivePlane| . PROJPL)
(|ProjectivePlaneOverPseudoAlgebraicClosureOfFiniteField| . PROJPLPS)
(|ProjectiveSpace| . PROJSP)
(|PseudoAlgebraicClosureOfAlgExtOfRationalNumber| . PACEXT)
(|QuadraticForm| . QFORM)
(|QuasiComponentPackage| . QCMPACK)
(|Quaternion| . QUAT)
(|QuaternionCategoryFunctions2| . QUATCT2)
(|QueryEquation| . QEQUAT)
(|Queue| . QUEUE)
(|QuotientFieldCategoryFunctions2| . QFCAT2)
(|RadicalEigenPackage| . REP)
(|RadicalSolvePackage| . SOLVERAD)
(|RadixExpansion| . RADIX)
(|RadixUtilities| . RADUTIL)
(|RandomNumberSource| . RANDSRC)
```

```
(|RationalFunction| . RF)
(|RationalFunctionDefiniteIntegration| . DEFINTRF)
(|RationalFunctionFactor| . RFFACT)
(|RationalFunctionFactorizer| . RFFACTOR)
(|RationalFunctionIntegration| . INTRF)
(|RationalFunctionLimitPackage| . LIMITRF)
(|RationalFunctionSign| . SIGNRF)
(|RationalFunctionSum| . SUMRF)
(|RationalRetractions| . RATRET)
(|RealClosure| . RECLOS)
(|RealPolynomialUtilitiesPackage| . POLUTIL)
(|RealZeroPackage| . REAL0)
(|RealZeroPackageQ| . REAL0Q)
(|RecurrenceOperator| . RECOP)
(|RectangularMatrixCategoryFunctions2| . RMCAT2)
(|RegularSetDecompositionPackage| . RSDCMPK)
(|RegularTriangularSet| . REGSET)
(|RegularTriangularSetGcdPackage| . RSETGCD)
(|RepresentationPackage1| . REP1)
(|RepresentationPackage2| . REP2)
(|ResolveLatticeCompletion| . RESLATC)
(|RewriteRule| . RULE)
(|RightOpenIntervalRootCharacterization| . ROIRC)
(|RomanNumeral| . ROMAN)
(|RootsFindingPackage| . RFP)
(|Ruleset| . RULESET)
(|ScriptFormulaFormat| . FORMULA)
(|ScriptFormulaFormat1| . FORMULA1)
(|Segment| . SEG)
(|SegmentBinding| . SEGBIND)
(|SegmentBindingFunctions2| . SEGBIND2)
(|SegmentFunctions2| . SEG2)
(|Set| . SET)
(|SimpleAlgebraicExtensionAlgFactor| . SAEFACT)
(|SimplifyAlgebraicNumberConvertPackage| . SIMPAN)
(|SingleInteger| . SINT)
(|SmithNormalForm| . SMITH)
(|SparseUnivariatePolynomialExpressions| . SUPEXPR)
(|SparseUnivariatePolynomialFunctions2| . SUP2)
(|SpecialOutputPackage| . SPECOUT)
(|SquareFreeRegularSetDecompositionPackage| . SRDCMPK)
(|SquareFreeRegularTriangularSet| . SREGSET)
(|SquareFreeRegularTriangularSetGcdPackage| . SFRGCD)
(|SquareFreeQuasiComponentPackage| . SFQCMPK)
(|Stack| . STACK)
(|Stream| . STREAM)
(|StreamFunctions1| . STREAM1)
(|StreamFunctions2| . STREAM2)
(|StreamFunctions3| . STREAM3)
(|StreamTensor| . STNSR)
```

```
  (|StochasticDifferential| . SD)
  (|String| . STRING)
  (|SturmHabichtPackage| . SHP)
  (|Symbol| . SYMBOL)
  (|SymmetricGroupCombinatoricFunctions| . SGCF)
  (|SystemSolvePackage| . SYSSOLP)
  (|SAERationalFunctionAlgFactor| . SAERFFC)
  (|Tableau| . TABLEAU)
  (|TaylorSeries| . TS)
  (|TaylorSolve| . UTSSOL)
  (|TexFormat| . TEX)
  (|TexFormat1| . TEX1)
  (|TextFile| . TEXTFILE)
  (|ThreeDimensionalViewport| . VIEW3D)
  (|ThreeSpace| . SPACE3)
  (|Timer|  . TIMER)
  (|TopLevelDrawFunctions| . DRAW)
  (|TopLevelDrawFunctionsForAlgebraicCurves| . DRAWCURV)
  (|TopLevelDrawFunctionsForCompiledFunctions| . DRAWCFUN)
  (|TopLevelDrawFunctionsForPoints| .  DRAWPT )
  (|TopLevelThreeSpace| . TOPSP)
  (|TranscendentalManipulations| . TRMANIP)
  (|TransSolvePackage| . SOLVETRA)
  (|Tree| . TREE)
  (|TrigonometricManipulations| . TRIGMNIP)
  (|UnivariateLaurentSeriesFunctions2| . ULS2)
  (|UnivariateFormalPowerSeries| . UFPS)
  (|UnivariateFormalPowerSeriesFunctions| . UFPS1)
  (|UnivariatePolynomial| . UP)
  (|UnivariatePolynomialCategoryFunctions2| . UPOLYC2)
  (|UnivariatePolynomialCommonDenominator| . UPCDEN)
  (|UnivariatePolynomialFunctions2| . UP2)
  (|UnivariatePolynomialMultiplicationPackage| . UPMP)
  (|UnivariateTaylorSeriesCZero| . UTSZ)
  (|UnivariatePuiseuxSeriesFunctions2| . UPXS2)
  (|UnivariateTaylorSeriesFunctions2| . UTS2)
  (|UniversalSegment| . UNISEG)
  (|UniversalSegmentFunctions2| . UNISEG2)
  (|UserDefinedVariableOrdering| . UDVO)
  (|U32Vector| . U32VEC)
  (|Vector| . VECTOR)
  (|VectorFunctions2| . VECTOR2)
  (|ViewDefaultsPackage| . VIEWDEF)
  (|Void| . VOID)
  (|WuWenTsunTriangularSet| . WUTSET))
(|naglink|
  (|Asp1| . ASP1)
  (|Asp4| . ASP4)
  (|Asp6| . ASP6)
  (|Asp7| . ASP7)
```

```
(|Asp8| . ASP8)
(|Asp9| . ASP9)
(|Asp10| . ASP10)
(|Asp12| . ASP12)
(|Asp19| . ASP19)
(|Asp20| . ASP20)
(|Asp24| . ASP24)
(|Asp27| . ASP27)
(|Asp28| . ASP28)
(|Asp29| . ASP29)
(|Asp30| . ASP30)
(|Asp31| . ASP31)
(|Asp33| . ASP33)
(|Asp34| . ASP34)
(|Asp35| . ASP35)
(|Asp41| . ASP41)
(|Asp42| . ASP42)
(|Asp49| . ASP49)
(|Asp50| . ASP50)
(|Asp55| . ASP55)
(|Asp73| . ASP73)
(|Asp74| . ASP74)
(|Asp77| . ASP77)
(|Asp78| . ASP78)
(|Asp80| . ASP80)
(|FortranCode| . FC)
(|FortranCodePackage1| . FCPAK1)
(|FortranExpression| . FEXPR)
(|FortranMachineTypeCategory| . FMTC)
(|FortranMatrixCategory| . FMC)
(|FortranMatrixFunctionCategory| . FMFUN)
(|FortranOutputStackPackage| . FOP)
(|FortranPackage| . FORT)
(|FortranProgramCategory| . FORTCAT)
(|FortranProgram| . FORTRAN)
(|FortranFunctionCategory| . FORTFN)
(|FortranScalarType| . FST)
(|FortranType| . FT)
(|FortranTemplate| . FTEM)
(|FortranVectorFunctionCategory| . FVFUN)
(|FortranVectorCategory| . FVC)
(|MachineComplex| . MCMPLX)
(|MachineFloat| . MFLOAT)
(|MachineInteger| . MINT)
(|MultiVariableCalculusFunctions| . MCALCFN)
(|NagDiscreteFourierTransformInterfacePackage| . NAGDIS)
(|NagEigenInterfacePackage| . NAGEIG)
(|NAGLinkSupportPackage| . NAGSP)
(|NagOptimisationInterfacePackage| . NAGOPT)
(|NagQuadratureInterfacePackage| . NAGQUA)
```

```
  (|NagResultChecks| . NAGRES)
  (|NagSpecialFunctionsInterfacePackage| . NAGSPE)
  (|NagPolynomialRootsPackage| . NAGC02)
  (|NagRootFindingPackage| . NAGC05)
  (|NagSeriesSummationPackage| . NAGC06)
  (|NagIntegrationPackage| . NAGD01)
  (|NagOrdinaryDifferentialEquationsPackage| . NAGD02)
  (|NagPartialDifferentialEquationsPackage| . NAGD03)
  (|NagInterpolationPackage| . NAGE01)
  (|NagFittingPackage| . NAGE02)
  (|NagOptimisationPackage| . NAGE04)
  (|NagMatrixOperationsPackage| . NAGF01)
  (|NagEigenPackage| . NAGF02)
  (|NagLinearEquationSolvingPackage| . NAGF04)
  (|NagLapack| . NAGF07)
  (|NagSpecialFunctionsPackage| . NAGS)
  (|PackedHermitianSequence| . PACKED)
  (|Result| . RESULT)
  (|SimpleFortranProgram| . SFORT)
  (|Switch| . SWITCH)
  (|SymbolTable| . SYMTAB)
  (|TemplateUtilities| . TEMUTL)
  (|TheSymbolTable| . SYMS)
  (|ThreeDimensionalMatrix| . M3D))
 (|anna|
  (|AnnaNumericalIntegrationPackage| . INTPACK)
  (|AnnaNumericalOptimizationPackage| . OPTPACK)
  (|AnnaOrdinaryDifferentialEquationPackage| . ODEPACK)
  (|AnnaPartialDifferentialEquationPackage| . PDEPACK)
  (|AttributeButtons| . ATTRBUT)
  (|BasicFunctions| . BFUNCT)
  (|d01ajfAnnaType| . D01AJFA)
  (|d01akfAnnaType| . D01AKFA)
  (|d01alfAnnaType| . D01ALFA)
  (|d01amfAnnaType| . D01AMFA)
  (|d01anfAnnaType| . D01ANFA)
  (|d01apfAnnaType| . D01APFA)
  (|d01aqfAnnaType| . D01AQFA)
  (|d01asfAnnaType| . D01ASFA)
  (|d01fcfAnnaType| . D01FCFA)
  (|d01gbfAnnaType| . D01GBFA)
  (|d01AgentsPackage| . D01AGNT)
  (|d01TransformFunctionType| . D01TRNS)
  (|d01WeightsPackage| . D01WGTS)
  (|d02AgentsPackage| . D02AGNT)
  (|d02bbfAnnaType| . D02BBFA)
  (|d02bhfAnnaType| . D02BHFA)
  (|d02cjfAnnaType| . D02CJFA)
  (|d02ejfAnnaType| . D02EJFA)
  (|d03AgentsPackage| . D03AGNT)
```

```
  (|d03eefAnnaType| . D03EEFA)
  (|d03fafAnnaType| . D03FAFA)
  (|e04AgentsPackage| . E04AGNT)
  (|e04dgfAnnaType| . E04DGFA)
  (|e04fdfAnnaType| . E04FDFA)
  (|e04gcfAnnaType| . E04GCFA)
  (|e04jafAnnaType| . E04JAFA)
  (|e04mbfAnnaType| . E04MBFA)
  (|e04nafAnnaType| . E04NAFA)
  (|e04ucfAnnaType| . E04UCFA)
  (|ExpertSystemContinuityPackage| . ESCONT)
  (|ExpertSystemContinuityPackage1| . ESCONT1)
  (|ExpertSystemToolsPackage| . ESTOOLS)
  (|ExpertSystemToolsPackage1| . ESTOOLS1)
  (|ExpertSystemToolsPackage2| . ESTOOLS2)
  (|NumericalIntegrationCategory| . NUMINT)
  (|NumericalIntegrationProblem| . NIPROB)
  (|NumericalODEProblem| . ODEPROB)
  (|NumericalOptimizationCategory| . OPTCAT)
  (|NumericalOptimizationProblem| . OPTPROB)
  (|NumericalPDEProblem| . PDEPROB)
  (|ODEIntensityFunctionsTable| . ODEIFTBL)
  (|IntegrationFunctionsTable| . INTFTBL)
  (|OrdinaryDifferentialEquationsSolverCategory| . ODECAT)
  (|PartialDifferentialEquationsSolverCategory| . PDECAT)
  (|RoutinesTable| . ROUTINE))
(|categories|
  (|AbelianGroup| . ABELGRP)
  (|AbelianMonoid| . ABELMON)
  (|AbelianMonoidRing| . AMR)
  (|AbelianSemiGroup| . ABELSG)
  (|AffineSpaceCategory| . AFSPCAT)
  (|Aggregate| . AGG)
  (|Algebra| . ALGEBRA)
  (|AlgebraicallyClosedField| . ACF)
  (|AlgebraicallyClosedFunctionSpace| . ACFS)
  (|ArcHyperbolicFunctionCategory| . AHYP)
  (|ArcTrigonometricFunctionCategory| . ATRIG)
  (|AssociationListAggregate| . ALAGG)
  (|AttributeRegistry| . ATTREG)
  (|BagAggregate| . BGAGG)
  (|BasicType| . BASTYPE)
  (|BiModule| . BMODULE)
  (|BinaryRecursiveAggregate| . BRAGG)
  (|BinaryTreeCategory| . BTCAT)
  (|BitAggregate| . BTAGG)
  (|BlowUpMethodCategory| . BLMETCT)
  (|CachableSet| . CACHSET)
  (|CancellationAbelianMonoid| . CABMON)
  (|CharacteristicNonZero| . CHARNZ)
```

```
(|CharacteristicZero| . CHARZ)
(|CoercibleTo| . KOERCE)
(|Collection| . CLAGG)
(|CombinatorialFunctionCategory| . CFCAT)
(|CombinatorialOpsCategory| . COMBOPC)
(|CommutativeRing| . COMRING)
(|ComplexCategory| . COMPCAT)
(|ConvertibleTo| . KONVERT)
(|DequeueAggregate| . DQAGG)
(|DesingTreeCategory| . DSTRCAT)
(|Dictionary| . DIAGG)
(|DictionaryOperations| . DIOPS)
(|DifferentialExtension| . DIFEXT)
(|DifferentialPolynomialCategory| . DPOLCAT)
(|DifferentialRing| . DIFRING)
(|DifferentialVariableCategory| . DVARCAT)
(|DirectProductCategory| . DIRPCAT)
(|DivisionRing| . DIVRING)
(|DivisorCategory| . DIVCAT)
(|DoublyLinkedAggregate| . DLAGG)
(|ElementaryFunctionCategory| . ELEMFUN)
(|Eltable| . ELTAB)
(|EltableAggregate| . ELTAGG)
(|EntireRing| . ENTIRER)
(|EuclideanDomain| . EUCDOM)
(|Evalable| . EVALAB)
(|ExpressionSpace| . ES)
(|ExtensibleLinearAggregate| . ELAGG)
(|ExtensionField| . XF)
(|Field| . FIELD)
(|FieldOfPrimeCharacteristic| . FPC)
(|Finite| . FINITE)
(|FileCategory| . FILECAT)
(|FileNameCategory| . FNCAT)
(|FiniteAbelianMonoidRing| . FAMR)
(|FiniteAlgebraicExtensionField| . FAXF)
(|FiniteDivisorCategory| . FDIVCAT)
(|FiniteFieldCategory| . FFIELDC)
(|FiniteLinearAggregate| . FLAGG)
(|FiniteRankNonAssociativeAlgebra| . FINAALG)
(|FiniteRankAlgebra| . FINRALG)
(|FiniteSetAggregate| . FSAGG)
(|FloatingPointSystem| . FPS)
(|FramedAlgebra| . FRAMALG)
(|FramedNonAssociativeAlgebra| . FRNAALG)
(|FramedNonAssociativeAlgebraFunctions2| . FRNAAF2)
(|FreeAbelianMonoidCategory| . FAMONC)
(|FreeLieAlgebra| . FLALG)
(|FreeModuleCat| . FMCAT)
(|FullyEvalableOver| . FEVALAB)
```

```
(|FullyLinearlyExplicitRingOver| . FLINEXP)
(|FullyPatternMatchable| . FPATMAB)
(|FullyRetractableTo| . FRETRCT)
(|FunctionFieldCategory| . FFCAT)
(|FunctionSpace| . FS)
(|GcdDomain| . GCDDOM)
(|GradedAlgebra| . GRALG)
(|GradedModule| . GRMOD)
(|Group| . GROUP)
(|HomogeneousAggregate| . HOAGG)
(|HyperbolicFunctionCategory| . HYPCAT)
(|IndexedAggregate| . IXAGG)
(|IndexedDirectProductCategory| . IDPC)
(|InfinitlyClosePointCategory| . INFCLCT)
(|InnerEvalable| . IEVALAB)
(|IntegerNumberSystem| . INS)
(|IntegralDomain| . INTDOM)
(|IntervalCategory| . INTCAT)
(|KeyedDictionary| . KDAGG)
(|LazyStreamAggregate| . LZSTAGG)
(|LeftAlgebra| . LALG)
(|LeftModule| . LMODULE)
(|LieAlgebra| . LIECAT)
(|LinearAggregate| . LNAGG)
(|LinearlyExplicitRingOver| . LINEXP)
(|LinearOrdinaryDifferentialOperatorCategory| . LODOCAT)
(|LiouvillianFunctionCategory| . LFCAT)
(|ListAggregate| . LSAGG)
(|LocalPowerSeriesCategory| . LOCPOWC)
(|Logic| . LOGIC)
(|MatrixCategory| . MATCAT)
(|Module| . MODULE)
(|Monad| . MONAD)
(|MonadWithUnit| . MONADWU)
(|Monoid| . MONOID)
(|MonogenicAlgebra| . MONOGEN)
(|MonogenicLinearOperator| . MLO)
(|MultiDictionary| . MDAGG)
(|MultisetAggregate| . MSETAGG)
(|MultivariateTaylorSeriesCategory| . MTSCAT)
(|NonAssociativeAlgebra| . NAALG)
(|NonAssociativeRing| . NASRING)
(|NonAssociativeRng| . NARNG)
(|NormalizedTriangularSetCategory| . NTSCAT)
(|Object| . OBJECT)
(|OctonionCategory| . OC)
(|OneDimensionalArrayAggregate| . A1AGG)
(|OpenMath| . OM)
(|OrderedAbelianGroup| . OAGROUP)
(|OrderedAbelianMonoid| . OAMON)
```

```
(|OrderedAbelianMonoidSup| . OAMONS)
(|OrderedAbelianSemiGroup| . OASGP)
(|OrderedCancellationAbelianMonoid| . OCAMON)
(|OrderedFinite| . ORDFIN)
(|OrderedIntegralDomain| . OINTDOM)
(|OrderedMonoid| . ORDMON)
(|OrderedMultisetAggregate| . OMSAGG)
(|OrderedRing| . ORDRING)
(|OrderedSet| . ORDSET)
(|PAdicIntegerCategory| . PADICCT)
(|PartialDifferentialRing| . PDRING)
(|PartialTranscendentalFunctions| . PTRANFN)
(|Patternable| . PATAB)
(|PatternMatchable| . PATMAB)
(|PermutationCategory| . PERMCAT)
(|PlacesCategory| . PLACESC)
(|PlottablePlaneCurveCategory| . PPCURVE)
(|PlottableSpaceCurveCategory| . PSCURVE)
(|PointCategory| . PTCAT)
(|PolynomialCategory| . POLYCAT)
(|PolynomialFactorizationExplicit| . PFECAT)
(|PolynomialSetCategory| . PSETCAT)
(|PowerSeriesCategory| . PSCAT)
(|PrimitiveFunctionCategory| . PRIMCAT)
(|PrincipalIdealDomain| . PID)
(|PriorityQueueAggregate| . PRQAGG)
(|ProjectiveSpaceCategory| . PRSPCAT)
(|PseudoAlgebraicClosureofAlgExtOfRationalNumberCategory| . PACEXTC)
(|PseudoAlgebraicClosureOfFiniteField| . PACOFF)
(|PseudoAlgebraicClosureOfFiniteFieldCategory| . PACFFC)
(|PseudoAlgebraicClosureOfPerfectFieldCategory| . PACPERC)
(|PseudoAlgebraicClosureOfRationalNumber| . PACRAT)
(|PseudoAlgebraicClosureOfRationalNumberCategory| . PACRATC)
(|QuaternionCategory| . QUATCAT)
(|QueueAggregate| . QUAGG)
(|QuotientFieldCategory| . QFCAT)
(|RadicalCategory| . RADCAT)
(|RealClosedField| . RCFIELD)
(|RealConstant| . REAL)
(|RealNumberSystem| . RNS)
(|RealRootCharacterizationCategory| . RRCC)
(|RectangularMatrixCategory| . RMATCAT)
(|RecursiveAggregate| . RCAGG)
(|RecursivePolynomialCategory| . RPOLCAT)
(|RegularChain| . RGCHAIN)
(|RegularTriangularSetCategory| . RSETCAT)
(|RetractableTo| . RETRACT)
(|RightModule| . RMODULE)
(|Ring| . RING)
(|Rng| . RNG)
```

```
  (|SegmentCategory| . SEGCAT)
  (|SegmentExpansionCategory| . SEGXCAT)
  (|SemiGroup| . SGROUP)
  (|SetAggregate| . SETAGG)
  (|SetCategory| . SETCAT)
  (|SetCategoryWithDegree| . SETCATD)
  (|SExpressionCategory| . SEXCAT)
  (|SpecialFunctionCategory| . SPFCAT)
  (|SquareFreeNormalizedTriangularSetCategory| . SNTSCAT)
  (|SquareFreeRegularTriangularSetCategory| . SFRTCAT)
  (|SquareMatrixCategory| . SMATCAT)
  (|StackAggregate| . SKAGG)
  (|StepThrough| . STEP)
  (|StreamAggregate| . STAGG)
  (|StringAggregate| . SRAGG)
  (|StringCategory| . STRICAT)
  (|StructuralConstantsPackage| . SCPKG)
  (|TableAggregate| . TBAGG)
  (|ThreeSpaceCategory| . SPACEC)
  (|TranscendentalFunctionCategory| . TRANFUN)
  (|TriangularSetCategory| . TSETCAT)
  (|TrigonometricFunctionCategory| . TRIGCAT)
  (|TwoDimensionalArrayCategory| . ARR2CAT)
  (|Type| . TYPE)
  (|UnaryRecursiveAggregate| . URAGG)
  (|UniqueFactorizationDomain| . UFD)
  (|UnivariateLaurentSeriesCategory| . ULSCAT)
  (|UnivariateLaurentSeriesConstructorCategory| . ULSCCAT)
  (|UnivariatePolynomialCategory| . UPOLYC)
  (|UnivariatePowerSeriesCategory| . UPSCAT)
  (|UnivariatePuiseuxSeriesCategory| . UPXSCAT)
  (|UnivariatePuiseuxSeriesConstructorCategory| . UPXSCCA)
  (|UnivariateSkewPolynomialCategory| . OREPCAT)
  (|UnivariateTaylorSeriesCategory| . UTSCAT)
  (|VectorCategory| . VECTCAT)
  (|VectorSpace| . VSPACE)
  (|XAlgebra| . XALG)
  (|XFreeAlgebra| . XFALG)
  (|XPolynomialsCat| . XPOLYC)
  (|ZeroDimensionalSolvePackage| . ZDSOLVE))
 (|Hidden|
  (|AlgebraicFunction| . AF)
  (|AlgebraicFunctionField| . ALGFF)
  (|AlgebraicHermiteIntegration| . INTHERAL)
  (|AlgebraicIntegrate| . INTALG)
  (|AlgebraicIntegration| . INTAF)
  (|AnonymousFunction| . ANON)
  (|AntiSymm| . ANTISYM)
  (|ApplyRules| . APPRULE)
  (|ApplyUnivariateSkewPolynomial| . APPLYORE)
```

```
(|ArrayStack| . ASTACK)
(|AssociatedEquations| . ASSOCEQ)
(|AssociationList| . ALIST)
(|Automorphism| . AUTOMOR)
(|BalancedFactorisation| . BALFACT)
(|BalancedPAdicInteger| . BPADIC)
(|BalancedPAdicRational| . BPADICRT)
(|BezoutMatrix| . BEZOUT)
(|BoundIntegerRoots| . BOUNDZRO)
(|BrillhartTests| . BRILL)
(|ChangeOfVariable| . CHVAR)
(|CharacteristicPolynomialInMonogenicalAlgebra| . CPIMA)
(|ChineseRemainderToolsForIntegralBases| . IBACHIN)
(|CoerceVectorMatrixPackage| . CVMP)
(|CombinatorialFunction| . COMBF)
(|CommonOperators| . COMMONOP)
(|CommuteUnivariatePolynomialCategory| . COMMUPC)
(|ComplexIntegerSolveLinearPolynomialEquation| . CINTSLPE)
(|ComplexPattern| . COMPLPAT)
(|ComplexPatternMatch| . CPMATCH)
(|ComplexRootFindingPackage| . CRFP)
(|ConstantLODE| . ODECONST)
(|CyclicStreamTools| . CSTTOOLS)
(|CyclotomicPolynomialPackage| . CYCLOTOM)
(|DefiniteIntegrationTools| . DFINTTLS)
(|DegreeReductionPackage| . DEGRED)
(|DeRhamComplex| . DERHAM)
(|DifferentialSparseMultivariatePolynomial| . DSMP)
(|DirectProduct| . DIRPROD)
(|DirectProductMatrixModule| . DPMM)
(|DirectProductModule| . DPMO)
(|DiscreteLogarithmPackage| . DLP)
(|DistributedMultivariatePolynomial| . DMP)
(|DoubleResultantPackage| . DBLRESP)
(|DrawOptionFunctions0| . DROPT0)
(|DrawOptionFunctions1| . DROPT1)
(|ElementaryFunction| . EF)
(|ElementaryFunctionsUnivariateLaurentSeries| . EFULS)
(|ElementaryFunctionsUnivariatePuiseuxSeries| . EFUPXS)
(|ElementaryIntegration| . INTEF)
(|ElementaryRischDE| . RDEEF)
(|ElementaryRischDESystem| . RDEEFS)
(|EllipticFunctionsUnivariateTaylorSeries| . ELFUTS)
(|EqTable| . EQTBL)
(|EuclideanModularRing| . EMR)
(|EvaluateCycleIndicators| . EVALCYC)
(|ExponentialExpansion| . EXPEXPAN)
(|ExponentialOfUnivariatePuiseuxSeries| . EXPUPXS)
(|ExpressionSpaceFunctions1| . ES1)
(|ExpressionTubePlot| . EXPRTUBE)
```

```
(|ExtAlgBasis| . EAB)
(|FactoredFunctions| . FACTFUNC)
(|FactoredFunctionUtilities| . FRUTIL)
(|FactoringUtilities| . FACUTIL)
(|FGLMIfCanPackage| . FGLMICPK)
(|FindOrderFinite| . FORDER)
(|FiniteDivisor| . FDIV)
(|FiniteFieldCyclicGroupExtension| . FFCGX)
(|FiniteFieldCyclicGroupExtensionByPolynomial| . FFCGP)
(|FiniteFieldExtension| . FFX)
(|FiniteFieldExtensionByPolynomial| . FFP)
(|FiniteFieldFunctions| . FFF)
(|FiniteFieldNormalBasisExtension| . FFNBX)
(|FiniteFieldNormalBasisExtensionByPolynomial| . FFNBP)
(|FiniteFieldPolynomialPackage| . FFPOLY)
(|FiniteFieldSolveLinearPolynomialEquation| . FFSLPE)
(|FormalFraction| . FORMAL)
(|FourierComponent| . FCOMP)
(|FractionalIdeal| . FRIDEAL)
(|FramedModule| . FRMOD)
(|FreeAbelianGroup| . FAGROUP)
(|FreeAbelianMonoid| . FAMONOID)
(|FreeGroup| . FGROUP)
(|FreeModule| . FM)
(|FreeModule1| . FM1)
(|FreeMonoid| . FMONOID)
(|FunctionalSpecialFunction| . FSPECF)
(|FunctionCalled| . FUNCTION)
(|FunctionFieldIntegralBasis| . FFINTBAS)
(|FunctionSpaceReduce| . FSRED)
(|FunctionSpaceToUnivariatePowerSeries| . FS2UPS)
(|FunctionSpaceToExponentialExpansion| . FS2EXPXP)
(|FunctionSpaceUnivariatePolynomialFactor| . FSUPFACT)
(|GaloisGroupFactorizationUtilities| . GALFACTU)
(|GaloisGroupFactorizer| . GALFACT)
(|GaloisGroupPolynomialUtilities| . GALPOLYU)
(|GaloisGroupUtilities| . GALUTIL)
(|GeneralHenselPackage| . GHENSEL)
(|GeneralDistributedMultivariatePolynomial| . GDMP)
(|GeneralPolynomialGcdPackage| . GENPGCD)
(|GeneralSparseTable| . GSTBL)
(|GenericNonAssociativeAlgebra| . GCNAALG)
(|GenExEuclid| . GENEEZ)
(|GeneralizedMultivariateFactorize| . GENMFACT)
(|GeneralModulePolynomial| . GMODPOL)
(|GeneralPolynomialSet| . GPOLSET)
(|GeneralTriangularSet| . GTSET)
(|GenUFactorize| . GENUFACT)
(|GenusZeroIntegration| . INTG0)
(|GosperSummationMethod| . GOSPER)
```

```
(|GraphImage| . GRIMAGE)
(|GrayCode| . GRAY)
(|GroebnerInternalPackage| . GBINTERN)
(|GroebnerSolve| . GROEBSOL)
(|GuessOptionFunctions0| . GOPT0)
(|HashTable| . HASHTBL)
(|Heap| . HEAP)
(|HeuGcd| . HEUGCD)
(|HomogeneousDistributedMultivariatePolynomial| . HDMP)
(|HyperellipticFiniteDivisor| . HELLFDIV)
(|IncrementingMaps| . INCRMAPS)
(|IndexedBits| . IBITS)
(|IndexedDirectProductAbelianGroup| . IDPAG)
(|IndexedDirectProductAbelianMonoid| . IDPAM)
(|IndexedDirectProductObject| . IDPO)
(|IndexedDirectProductOrderedAbelianMonoid| . IDPOAM)
(|IndexedDirectProductOrderedAbelianMonoidSup| . IDPOAMS)
(|IndexedExponents| . INDE)
(|IndexedFlexibleArray| . IFARRAY)
(|IndexedList| . ILIST)
(|IndexedMatrix| . IMATRIX)
(|IndexedOneDimensionalArray| . IARRAY1)
(|IndexedString| . ISTRING)
(|IndexedTwoDimensionalArray| . IARRAY2)
(|IndexedVector| . IVECTOR)
(|InnerAlgFactor| . IALGFACT)
(|InnerAlgebraicNumber| . IAN)
(|InnerCommonDenominator| . ICDEN)
(|InnerFiniteField| . IFF)
(|InnerFreeAbelianMonoid| . IFAMON)
(|InnerIndexedTwoDimensionalArray| . IIARRAY2)
(|InnerMatrixLinearAlgebraFunctions| . IMATLIN)
(|InnerMatrixQuotientFieldFunctions| . IMATQF)
(|InnerModularGcd| . INMODGCD)
(|InnerMultFact| . INNMFACT)
(|InnerNormalBasisFieldFunctions| . INBFF)
(|InnerNumericEigenPackage| . INEP)
(|InnerNumericFloatSolvePackage| . INFSP)
(|InnerPAdicInteger| . IPADIC)
(|InnerPolySign| . INPSIGN)
(|InnerPolySum| . ISUMP)
(|InnerPrimeField| . IPF)
(|InnerSparseUnivariatePowerSeries| . ISUPS)
(|InnerTable| . INTABL)
(|InnerTaylorSeries| . ITAYLOR)
(|InnerTrigonometricManipulations| . ITRIGMNP)
(|InputForm| . INFORM)
(|InputFormFunctions1| . INFORM1)
(|IntegerBits| . INTBIT)
(|IntegerFactorizationPackage| . INTFACT)
```

```
(|IntegerMod| . ZMOD)
(|IntegerSolveLinearPolynomialEquation| . INTSLPE)
(|IntegralBasisPolynomialTools| . IBPTOOLS)
(|IntegralBasisTools| . IBATOOL)
(|IntegrationResult| . IR)
(|IntegrationTools| . INTTOOLS)
(|InternalPrintPackage| . IPRNTPK)
(|InternalRationalUnivariateRepresentationPackage| . IRURPK)
(|IrredPolyOverFiniteField| . IRREDFFX)
(|Kernel| . KERNEL)
(|Kovacic| . KOVACIC)
(|LaurentPolynomial| . LAUPOL)
(|LeadingCoefDetermination| . LEADCDET)
(|LexTriangularPackage| . LEXTRIPK)
(|LieExponentials| . LEXP)
(|LiePolynomial| . LPOLY)
(|LinearDependence| . LINDEP)
(|LinearOrdinaryDifferentialOperatorFactorizer| . LODOF)
(|LinearOrdinaryDifferentialOperator1| . LODO1)
(|LinearOrdinaryDifferentialOperator2| . LODO2)
(|LinearOrdinaryDifferentialOperatorsOps| . LODOOPS)
(|LinearPolynomialEquationByFractions| . LPEFRAC)
(|LinGroebnerPackage| . LGROBP)
(|LiouvillianFunction| . LF)
(|ListMonoidOps| . LMOPS)
(|ListMultiDictionary| . LMDICT)
(|LocalAlgebra| . LA)
(|Localize| . LO)
(|LyndonWord| . LWORD)
(|Magma| . MAGMA)
(|MakeBinaryCompiledFunction| . MKBCFUNC)
(|MakeCachableSet| . MKCHSET)
(|MakeUnaryCompiledFunction| . MKUCFUNC)
(|MappingPackageInternalHacks1| . MAPHACK1)
(|MappingPackageInternalHacks2| . MAPHACK2)
(|MappingPackageInternalHacks3| . MAPHACK3)
(|MeshCreationRoutinesForThreeDimensions| . MESH)
(|ModMonic| . MODMON)
(|ModularField| . MODFIELD)
(|ModularHermitianRowReduction| . MHROWRED)
(|ModularRing| . MODRING)
(|ModuleMonomial| . MODMONOM)
(|MoebiusTransform| . MOEBIUS)
(|MonoidRing| . MRING)
(|MonomialExtensionTools| . MONOTOOL)
(|MPolyCatPolyFactorizer| . MPCPF)
(|MPolyCatFunctions3| . MPC3)
(|MRationalFactorize| . MRATFAC)
(|MultipleMap| . MMAP)
(|MultivariateLifting| . MLIFT)
```

```
(|MultivariateSquareFree| . MULTSQFR)
(|HomogeneousDirectProduct| . HDP)
(|NewSparseMultivariatePolynomial| . NSMP)
(|NewSparseUnivariatePolynomial| . NSUP)
(|NewSparseUnivariatePolynomialFunctions2| . NSUP2)
(|NonCommutativeOperatorDivision| . NCODIV)
(|NewtonInterpolation| . NEWTON)
(|None| . NONE)
(|NonLinearFirstOrderODESolver| . NODE1)
(|NonLinearSolvePackage| . NLINSOL)
(|NormRetractPackage| . NORMRETR)
(|NPCoef| . NPCOEF)
(|NumberFormats| . NUMFMT)
(|NumberFieldIntegralBasis| . NFINTBAS)
(|NumericTubePlot| . NUMTUBE)
(|ODEIntegration| . ODEINT)
(|ODETools| . ODETOOLS)
(|Operator| . OP)
(|OppositeMonogenicLinearOperator| . OMLO)
(|OrderedDirectProduct| . ODP)
(|OrderedFreeMonoid| . OFMONOID)
(|OrderedVariableList| . OVAR)
(|OrderingFunctions| . ORDFUNS)
(|OrderlyDifferentialPolynomial| . ODPOL)
(|OrderlyDifferentialVariable| . ODVAR)
(|OrdinaryWeightedPolynomials| . OWP)
(|OutputForm| . OUTFORM)
(|PadeApproximants| . PADE)
(|PAdicInteger| . PADIC)
(|PAdicRational| . PADICRAT)
(|PAdicRationalConstructor| . PADICRC)
(|PAdicWildFunctionFieldIntegralBasis| . PWFFINTB)
(|ParadoxicalCombinatorsForStreams| . YSTREAM)
(|ParametricLinearEquations| . PLEQN)
(|PartialFractionPackage| . PFRPAC)
(|Partition| . PRTITION)
(|Pattern| . PATTERN)
(|PatternFunctions1| . PATTERN1)
(|PatternMatchFunctionSpace| . PMFS)
(|PatternMatchIntegerNumberSystem| . PMINS)
(|PatternMatchIntegration| . INTPM)
(|PatternMatchKernel| . PMKERNEL)
(|PatternMatchListAggregate| . PMLSAGG)
(|PatternMatchListResult| . PATLRES)
(|PatternMatchPolynomialCategory| . PMPLCAT)
(|PatternMatchPushDown| . PMDOWN)
(|PatternMatchQuotientFieldCategory| . PMQFCAT)
(|PatternMatchResult| . PATRES)
(|PatternMatchSymbol| . PMSYM)
(|PatternMatchTools| . PMTOOLS)
```

```
(|PlaneAlgebraicCurvePlot| . ACPLOT)
(|Plot| . PLOT)
(|PlotFunctions1| . PLOT1)
(|PlotTools| . PLOTTOOL)
(|Plot3D| . PLOT3D)
(|PoincareBirkhoffWittLyndonBasis| . PBWLB)
(|Point| . POINT)
(|PointsOfFiniteOrder| . PFO)
(|PointsOfFiniteOrderRational| . PFOQ)
(|PointsOfFiniteOrderTools| . PFOTOOLS)
(|PointPackage| . PTPACK)
(|PolToPol| . POLTOPOL)
(|PolynomialCategoryLifting| . POLYLIFT)
(|PolynomialCategoryQuotientFunctions| . POLYCATQ)
(|PolynomialFactorizationByRecursion| . PFBR)
(|PolynomialFactorizationByRecursionUnivariate| . PFBRU)
(|PolynomialGcdPackage| . PGCD)
(|PolynomialInterpolation| . PINTERP)
(|PolynomialInterpolationAlgorithms| . PINTERPA)
(|PolynomialNumberTheoryFunctions| . PNTHEORY)
(|PolynomialRing| . PR)
(|PolynomialRoots| . POLYROOT)
(|PolynomialSetUtilitiesPackage| . PSETPK)
(|PolynomialSolveByFormulas| . SOLVEFOR)
(|PolynomialSquareFree| . PSQFR)
(|PrecomputedAssociatedEquations| . PREASSOC)
(|PrimitiveArray| . PRIMARR)
(|PrimitiveElement| . PRIMELT)
(|PrimitiveRatDE| . ODEPRIM)
(|PrimitiveRatRicDE| . ODEPRRIC)
(|Product| . PRODUCT)
(|PseudoRemainderSequence| . PRS)
(|PseudoLinearNormalForm| . PSEUDLIN)
(|PureAlgebraicIntegration| . INTPAF)
(|PureAlgebraicLODE| . ODEPAL)
(|PushVariables| . PUSHVAR)
(|QuasiAlgebraicSet| . QALGSET)
(|QuasiAlgebraicSet2| . QALGSET2)
(|RadicalFunctionField| . RADFF)
(|RandomDistributions| . RDIST)
(|RandomFloatDistributions| . RFDIST)
(|RandomIntegerDistributions| . RIDIST)
(|RationalFactorize| . RATFACT)
(|RationalIntegration| . INTRAT)
(|RationalInterpolation| . RINTERP)
(|RationalLODE| . ODERAT)
(|RationalRicDE| . ODERTRIC)
(|RationalUnivariateRepresentationPackage| . RURPK)
(|RealSolvePackage| . REALSOLV)
(|RectangularMatrix| . RMATRIX)
```

```
(|ReducedDivisor| . RDIV)
(|ReduceLODE| . ODERED)
(|ReductionOfOrder| . REDORDER)
(|Reference| . REF)
(|RepeatedDoubling| . REPDB)
(|RepeatedSquaring| . REPSQ)
(|ResidueRing| . RESRING)
(|RetractSolvePackage| . RETSOL)
(|RuleCalled| . RULECOLD)
(|SetOfMIntegersInOneToN| . SETMN)
(|SExpression| . SEX)
(|SExpressionOf| . SEXOF)
(|SequentialDifferentialPolynomial| . SDPOL)
(|SequentialDifferentialVariable| . SDVAR)
(|SimpleAlgebraicExtension| . SAE)
(|SingletonAsOrderedSet| . SAOS)
(|SortedCache| . SCACHE)
(|SortPackage| . SORTPAK)
(|SparseMultivariatePolynomial| . SMP)
(|SparseMultivariateTaylorSeries| . SMTS)
(|SparseTable| . STBL)
(|SparseUnivariatePolynomial| . SUP)
(|SparseUnivariateSkewPolynomial| . ORESUP)
(|SparseUnivariateLaurentSeries| . SULS)
(|SparseUnivariatePuiseuxSeries| . SUPXS)
(|SparseUnivariateTaylorSeries| . SUTS)
(|SplitHomogeneousDirectProduct| . SHDP)
(|SplittingNode| . SPLNODE)
(|SplittingTree| . SPLTREE)
(|SquareMatrix| . SQMATRIX)
(|Stack| . STACK)
(|StorageEfficientMatrixOperations| . MATSTOR)
(|StreamInfiniteProduct| . STINPROD)
(|StreamTaylorSeriesOperations| . STTAYLOR)
(|StreamTranscendentalFunctions| . STTF)
(|StreamTranscendentalFunctionsNonCommutative| . STTFNC)
(|StringTable| . STRTBL)
(|SubResultantPackage| . SUBRESP)
(|SubSpace| . SUBSPACE)
(|SubSpaceComponentProperty| . COMPPROP)
(|SuchThat| . SUCH)
(|SupFractionFactorizer| . SUPFRACF)
(|SymmetricFunctions| . SYMFUNC)
(|SymmetricPolynomial| . SYMPOLY)
(|SystemODESolver| . ODESYS)
(|Table| . TABLE)
(|TableauxBumpers| . TABLBUMP)
(|TabulatedComputationPackage| . TBCMPPK)
(|TangentExpansions| . TANEXP)
(|ToolsForSign| . TOOLSIGN)
```

```
  (|TranscendentalHermiteIntegration| . INTHERTR)
  (|TranscendentalIntegration| . INTTR)
  (|TranscendentalRischDE| . RDETR)
  (|TranscendentalRischDESystem| . RDETRS)
  (|TransSolvePackageService| . SOLVESER)
  (|TriangularMatrixOperations| . TRIMAT)
  (|TubePlot| . TUBE)
  (|TubePlotTools| . TUBETOOL)
  (|Tuple| . TUPLE)
  (|TwoDimensionalArray| . ARRAY2)
  (|TwoDimensionalPlotClipping| . CLIP)
  (|TwoDimensionalViewport| . VIEW2D)
  (|TwoFactorize| . TWOFACT)
  (|UnivariateFactorize| . UNIFACT)
  (|UnivariateLaurentSeries| . ULS)
  (|UnivariateLaurentSeriesConstructor| . ULSCONS)
  (|UnivariatePolynomialDecompositionPackage| . UPDECOMP)
  (|UnivariatePolynomialDivisionPackage| . UPDIVP)
  (|UnivariatePolynomialSquareFree| . UPSQFREE)
  (|UnivariatePuiseuxSeries| . UPXS)
  (|UnivariatePuiseuxSeriesConstructor| . UPXSCONS)
  (|UnivariatePuiseuxSeriesWithExponentialSingularity| . UPXSSING)
  (|UnivariateSkewPolynomial| . OREUP)
  (|UnivariateSkewPolynomialCategoryOps| . OREPCTO)
  (|UnivariateTaylorSeries| . UTS)
  (|UnivariateTaylorSeriesODESolver| . UTSODE)
  (|UserDefinedPartialOrdering| . UDPO)
  (|UTSodetools| . UTSODETL)
  (|Variable| . VARIABLE)
  (|ViewportPackage| . VIEW)
  (|WeierstrassPreparation| . WEIER)
  (|WeightedPolynomials| . WP)
  (|WildFunctionFieldIntegralBasis| . WFFINTBS)
  (|XDistributedPolynomial| . XDPOLY)
  (|XExponentialPackage| . XEXPPKG)
  (|XPBWPolynomial| . XPBWPOLY)
  (|XPolynomial| . XPOLY)
  (|XPolynomialRing| . XPR)
  (|XRecursivePolynomial| . XRPOLY))
 (|defaults|
  (|AbelianGroup&| . ABELGRP-)
  (|AbelianMonoid&| . ABELMON-)
  (|AbelianMonoidRing&| . AMR-)
  (|AbelianSemiGroup&| . ABELSG-)
  (|Aggregate&| . AGG-)
  (|Algebra&| . ALGEBRA-)
  (|AlgebraicallyClosedField&| . ACF-)
  (|AlgebraicallyClosedFunctionSpace&| . ACFS-)
  (|ArcTrigonometricFunctionCategory&| . ATRIG-)
  (|BagAggregate&| . BGAGG-)
```

```
(|BasicType&| . BASTYPE-)
(|BinaryRecursiveAggregate&| . BRAGG-)
(|BinaryTreeCategory&| . BTCAT-)
(|BitAggregate&| . BTAGG-)
(|Collection&| . CLAGG-)
(|ComplexCategory&| . COMPCAT-)
(|Dictionary&| . DIAGG-)
(|DictionaryOperations&| . DIOPS-)
(|DifferentialExtension&| . DIFEXT-)
(|DifferentialPolynomialCategory&| . DPOLCAT-)
(|DifferentialRing&| . DIFRING-)
(|DifferentialVariableCategory&| . DVARCAT-)
(|DirectProductCategory&| . DIRPCAT-)
(|DivisionRing&| . DIVRING-)
(|ElementaryFunctionCategory&| . ELEMFUN-)
(|EltableAggregate&| . ELTAGG-)
(|EuclideanDomain&| . EUCDOM-)
(|Evalable&| . EVALAB-)
(|ExpressionSpace&| . ES-)
(|ExtensibleLinearAggregate&| . ELAGG-)
(|ExtensionField&| . XF-)
(|Field&| . FIELD-)
(|FieldOfPrimeCharacteristic&| . FPC-)
(|FiniteAbelianMonoidRing&| . FAMR-)
(|FiniteAlgebraicExtensionField&| . FAXF-)
(|FiniteDivisorCategory&| . FDIVCAT-)
(|FiniteFieldCategory&| . FFIELDC-)
(|FiniteLinearAggregate&| . FLAGG-)
(|FiniteSetAggregate&| . FSAGG-)
(|FiniteRankAlgebra&| . FINRALG-)
(|FiniteRankNonAssociativeAlgebra&| . FINAALG-)
(|FloatingPointSystem&| . FPS-)
(|FramedAlgebra&| . FRAMALG-)
(|FramedNonAssociativeAlgebra&| . FRNAALG-)
(|FullyEvalableOver&| . FEVALAB-)
(|FullyLinearlyExplicitRingOver&| . FLINEXP-)
(|FullyRetractableTo&| . FRETRCT-)
(|FunctionFieldCategory&| . FFCAT-)
(|FunctionSpace&| . FS-)
(|GcdDomain&| . GCDDOM-)
(|GradedAlgebra&| . GRALG-)
(|GradedModule&| . GRMOD-)
(|Group&| . GROUP-)
(|HomogeneousAggregate&| . HOAGG-)
(|HyperbolicFunctionCategory&| . HYPCAT-)
(|IndexedAggregate&| . IXAGG-)
(|InnerEvalable&| . IEVALAB-)
(|IntegerNumberSystem&| . INS-)
(|IntegralDomain&| . INTDOM-)
(|KeyedDictionary&| . KDAGG-)
```

```
(|LazyStreamAggregate&| . LZSTAGG-)
(|LeftAlgebra&| . LALG-)
(|LieAlgebra&| . LIECAT-)
(|LinearAggregate&| . LNAGG-)
(|ListAggregate&| . LSAGG-)
(|Logic&| . LOGIC-)
(|LinearOrdinaryDifferentialOperatorCategory&| . LODOCAT-)
(|MatrixCategory&| . MATCAT-)
(|Module&| . MODULE-)
(|Monad&| . MONAD-)
(|MonadWithUnit&| . MONADWU-)
(|Monoid&| . MONOID-)
(|MonogenicAlgebra&| . MONOGEN-)
(|NonAssociativeAlgebra&| . NAALG-)
(|NonAssociativeRing&| . NASRING-)
(|NonAssociativeRng&| . NARNG-)
(|OctonionCategory&| . OC-)
(|OneDimensionalArrayAggregate&| . A1AGG-)
(|OrderedRing&| . ORDRING-)
(|OrderedSet&| . ORDSET-)
(|PartialDifferentialRing&| . PDRING-)
(|PolynomialCategory&| . POLYCAT-)
(|PolynomialFactorizationExplicit&| . PFECAT-)
(|PolynomialSetCategory&| . PSETCAT-)
(|PowerSeriesCategory&| . PSCAT-)
(|QuaternionCategory&| . QUATCAT-)
(|QuotientFieldCategory&| . QFCAT-)
(|RadicalCategory&| . RADCAT-)
(|RealClosedField&| . RCFIELD-)
(|RealNumberSystem&| . RNS-)
(|RealRootCharacterizationCategory&| . RRCC-)
(|RectangularMatrixCategory&| . RMATCAT-)
(|RecursiveAggregate&| . RCAGG-)
(|RecursivePolynomialCategory&| . RPOLCAT-)
(|RegularTriangularSetCategory&| . RSETCAT-)
(|RetractableTo&| . RETRACT-)
(|Ring&| . RING-)
(|SemiGroup&| . SGROUP-)
(|SetAggregate&| . SETAGG-)
(|SetCategory&| . SETCAT-)
(|SquareMatrixCategory&| . SMATCAT-)
(|StreamAggregate&| . STAGG-)
(|StringAggregate&| . SRAGG-)
(|TableAggregate&| . TBAGG-)
(|TranscendentalFunctionCategory&| . TRANFUN-)
(|TriangularSetCategory&| . TSETCAT-)
(|TrigonometricFunctionCategory&| . TRIGCAT-)
(|TwoDimensionalArrayCategory&| . ARR2CAT-)
(|UnaryRecursiveAggregate&| . URAGG-)
(|UniqueFactorizationDomain&| . UFD-)
```

```
   (|UnivariateLaurentSeriesConstructorCategory&| . ULSCCAT-)
   (|UnivariatePolynomialCategory&| . UPOLYC-)
   (|UnivariatePowerSeriesCategory&| . UPSCAT-)
   (|UnivariatePuiseuxSeriesConstructorCategory&| . UPXSCCA-)
   (|UnivariateSkewPolynomialCategory&| . OREPCAT-)
   (|UnivariateTaylorSeriesCategory&| . UTSCAT-)
   (|VectorCategory&| . VECTCAT-)
   (|VectorSpace&| . VSPACE-))))
```

———-

45.18.2 defvar $localExposureDataDefault

— **initvars** —

```
(defvar |$localExposureDataDefault|
  (vector
    ;;These groups will be exposed
   (list '|basic| '|categories| '|naglink| '|anna|)
    ;;These constructors will be explicitly exposed
   (list )
    ;;These constructors will be explicitly hidden
   (list )))
```

———-

45.18.3 defvar $localExposureData

— **initvars** —

```
(defvar |$localExposureData|  (copy-seq |$localExposureDataDefault|))
```

———-

45.19 Functions

45.19.1 The top level set expose command handler

[displayExposedGroups p696]
[sayMSG p329]

[displayExposedConstructors p696]
[displayHiddenConstructors p697]
[sayKeyedMsg p327]
[namestring p1034]
[pathname p1036]
[qcar p??]
[qcdr p??]
[selectOptionLC p475]
[setExposeAdd p689]
[setExposeDrop p693]
[setExpose p688]

— **defun setExpose** —

```
(defun |setExpose| (arg)
 "The top level set expose command handler"
 (let (fnargs fn)
   (cond
    ((eq arg '|%initialize%|))
    ((eq arg '|%display%|) "...")
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|displayExposedGroups|)
     (|sayMSG| " ")
     (|displayExposedConstructors|)
     (|sayMSG| " ")
     (|displayHiddenConstructors|)
     (|sayMSG| " "))
    ((and (consp arg)
          (progn (setq fn (qcar arg)) (setq fnargs (qcdr arg)) t)
          (setq fn (|selectOptionLC| fn '(|add| |drop|) nil)))
     (cond
      ((eq fn '|add|) (|setExposeAdd| fnargs))
      ((eq fn '|drop|) (|setExposeDrop| fnargs))
      (t nil)))
    (t (|setExpose| nil)))))
```

———-

45.19.2 The top level set expose add command handler

[centerAndHighlight p??]
[specialChar p970]
[displayExposedGroups p696]
[sayMSG p329]
[displayExposedConstructors p696]
[sayKeyedMsg p327]

[qcar p**??**]
[qcdr p**??**]
[selectOptionLC p475]
[setExposeAddGroup p690]
[setExposeAddConstr p692]
[setExposeAdd p689]
[$linelength p765]

— defun setExposeAdd —

```
(defun |setExposeAdd| (arg)
 "The top level set expose add command handler"
 (declare (special $linelength))
 (let (fnargs fn)
   (cond
    ((null arg)
     (|centerAndHighlight|
      '|The add Option| $linelength (|specialChar| '|hbar|))
     (|displayExposedGroups|)
     (|sayMSG| " ")
     (|displayExposedConstructors|)
     (|sayMSG| " ")
     (|sayKeyedMsg| 's2iz0049e nil))
    ((and (consp arg)
          (progn (setq fn (qcar arg)) (setq fnargs (qcdr arg)) t)
          (setq fn (|selectOptionLC| fn '(|group| |constructor|) nil)))
     (cond
      ((eq fn '|group|) (|setExposeAddGroup| fnargs))
      ((eq fn '|constructor|) (|setExposeAddConstr| fnargs))
      (t nil)))
    (t (|setExposeAdd| nil)))))
```

———-

45.19.3 Expose a group

Note that `$localExposureData` is a vector of lists. It consists of [exposed groups,exposed constructors,hidden constructors] [object2String p**??**]
[qcar p**??**]
[setelt p**??**]
[displayExposedGroups p696]
[sayMSG p329]
[displayExposedConstructors p696]
[displayHiddenConstructors p697]
[clearClams p**??**]
[getalist p**??**]

[sayKeyedMsg p327]
[member p1042]
[msort p**??**]
[centerAndHighlight p**??**]
[specialChar p970]
[namestring p1034]
[pathname p1036]
[sayAsManyPerLineAsPossible p**??**]
[$globalExposureGroupAlist p662]
[$localExposureData p688]
[$interpreterFrameName p**??**]
[$linelength p765]

— defun setExposeAddGroup —

```
(defun |setExposeAddGroup| (arg)
 "Expose a group"
 (declare (special |$globalExposureGroupAlist| |$localExposureData|
                   |$interpreterFrameName| $linelength))
 (if (null arg)
  (progn
   (|centerAndHighlight|
    '|The group Option| $linelength (|specialChar| '|hbar|))
   (|displayExposedGroups|)
   (|sayMSG| " ")
   (|sayAsManyPerLineAsPossible|
    (mapcar #'(lambda (x) (|object2String| (first x)))
      |$globalExposureGroupAlist|)))
  (dolist (x arg)
    (when (consp x) (setq x (qcar x)))
    (cond
     ((eq x '|all|)
      (setelt |$localExposureData| 0
        (mapcar #'first |$globalExposureGroupAlist|))
      (setelt |$localExposureData| 1 nil)
      (setelt |$localExposureData| 2 nil)
      (|displayExposedGroups|)
      (|sayMSG| " ")
      (|displayExposedConstructors|)
      (|sayMSG| " ")
      (|displayHiddenConstructors|)
      (|clearClams|))
     ((null (getalist |$globalExposureGroupAlist| x))
      (|sayKeyedMsg| 's2iz0049h (cons x nil)))
     ((|member| x (elt |$localExposureData| 0))
      (|sayKeyedMsg| 's2iz0049i (list x |$interpreterFrameName|)))
     (t
      (setelt |$localExposureData| 0
        (msort (cons x (elt |$localExposureData| 0))))
```

```
      (|sayKeyedMsg| 's2iz0049r (list x |$interpreterFrameName|))
      (|clearClams|))))))
```

———-

45.19.4 The top level set expose add constructor handler

[unabbrev p??]
[qcar p??]
[getdatabase p1001]
[sayKeyedMsg p327]
[member p1042]
[setelt p??]
[delete p??]
[msort p??]
[clearClams p??]
[centerAndHighlight p??]
[specialChar p970]
[displayExposedConstructors p696]
[$linelength p765]
[$localExposureData p688]
[$interpreterFrameName p??]

— defun setExposeAddConstr —

```
(defun |setExposeAddConstr| (arg)
 "The top level set expose add constructor handler"
 (declare (special $linelength |$localExposureData| |$interpreterFrameName|))
 (if (null arg)
   (progn
    (|centerAndHighlight|
     '|The constructor Option| $linelength (|specialChar| '|hbar|))
    (|displayExposedConstructors|))
   (dolist (x arg)
     (setq x (|unabbrev| x))
     (when (consp x) (setq x (qcar x)))
     (cond
      ((null (getdatabase x 'constructorkind))
       (|sayKeyedMsg| 's2iz0049j (list x)))
      ((|member| x (elt |$localExposureData| 1))
       (|sayKeyedMsg| 's2iz0049k (list x |$interpreterFrameName| )))
      (t
       (when (|member| x (elt |$localExposureData| 2))
         (setelt |$localExposureData| 2
          (|delete| x (elt |$localExposureData| 2))))
       (setelt |$localExposureData| 1
```

```
        (msort (cons x (elt |$localExposureData| 1))))
       (|clearClams|)
       (|sayKeyedMsg| 's2iz0049p (list x |$interpreterFrameName| )))))))
```

———-

45.19.5 The top level set expose drop handler

[centerAndHighlight p??]
[specialChar p970]
[displayHiddenConstructors p697]
[sayMSG p329]
[sayKeyedMsg p327]
[qcar p??]
[qcdr p??]
[selectOptionLC p475]
[setExposeDropGroup p694]
[setExposeDropConstr p695]
[setExposeDrop p693]
[$linelength p765]

— **defun setExposeDrop** —

```
(defun |setExposeDrop| (arg)
 "The top level set expose drop handler"
 (declare (special $linelength))
 (let (fnargs fn)
   (cond
    ((null arg)
     (|centerAndHighlight|
      '|The drop Option| $linelength (|specialChar| '|hbar|))
     (|displayHiddenConstructors|)
     (|sayMSG| " ")
     (|sayKeyedMsg| 's2iz0049f nil))
    ((and (consp arg)
          (progn (setq fn (qcar arg)) (setq fnargs (qcdr arg)) t)
          (setq fn (|selectOptionLC| fn '(|group| |constructor|) nil)))
     (cond
      ((eq fn '|group|) (|setExposeDropGroup| fnargs))
      ((eq fn '|constructor|) (|setExposeDropConstr| fnargs))
      (t nil)))
    (t (|setExposeDrop| nil)))))
```

———-

45.19.6 The top level set expose drop group handler

[qcar p**??**]
[setelt p**??**]
[displayExposedGroups p696]
[sayMSG p329]
[displayExposedConstructors p696]
[displayHiddenConstructors p697]
[clearClams p**??**]
[member p1042]
[delete p**??**]
[sayKeyedMsg p327]
[getalist p**??**]
[centerAndHighlight p**??**]
[specialChar p970]
[$linelength p765]
[$localExposureData p688]
[$interpreterFrameName p**??**]
[$globalExposureGroupAlist p662]

— defun setExposeDropGroup —

```
(defun |setExposeDropGroup| (arg)
 "The top level set expose drop group handler"
 (declare (special $linelength |$localExposureData| |$interpreterFrameName|
                   |$globalExposureGroupAlist|))
 (if (null arg)
  (progn
   (|centerAndHighlight|
     '|The group Option| $linelength (|specialChar| '|hbar|))
   (|sayKeyedMsg| 's2iz0049l nil)
   (|sayMSG| " ")
   (|displayExposedGroups|))
  (dolist (x arg)
    (when (consp x) (setq x (qcar x)))
    (cond
     ((eq x '|all|)
      (setelt |$localExposureData| 0 nil)
      (setelt |$localExposureData| 1 nil)
      (setelt |$localExposureData| 2 nil)
      (|displayExposedGroups|)
      (|sayMSG| " ")
      (|displayExposedConstructors|)
      (|sayMSG| " ")
      (|displayHiddenConstructors|)
      (|clearClams|))
     ((|member| x (elt |$localExposureData| 0))
      (setelt |$localExposureData| 0
```

```
      (|delete| x (elt |$localExposureData| 0)))
     (|clearClams|)
     (|sayKeyedMsg| 's2iz0049s (list x |$interpreterFrameName| )))
    ((getalist |$globalExposureGroupAlist| x)
     (|sayKeyedMsg| 's2iz0049i (list x |$interpreterFrameName| )))
    (t (|sayKeyedMsg| 's2iz0049h (list x )))))))
```

———-

45.19.7 The top level set expose drop constructor handler

[unabbrev p??]
[qcar p??]
[getdatabase p1001]
[sayKeyedMsg p327]
[member p1042]
[setelt p??]
[delete p??]
[msort p??]
[clearClams p??]
[centerAndHighlight p??]
[specialChar p970]
[sayMSG p329]
[displayExposedConstructors p696]
[displayHiddenConstructors p697]
[$linelength p765]
[$localExposureData p688]
[$interpreterFrameName p??]

— defun setExposeDropConstr —

```
(defun |setExposeDropConstr| (arg)
 "The top level set expose drop constructor handler"
 (declare (special $linelength |$localExposureData| |$interpreterFrameName|))
 (if (null arg)
  (progn
   (|centerAndHighlight|
    '|The constructor Option| $linelength (|specialChar| '|hbar|))
   (|sayKeyedMsg| 's2iz0049n nil)
   (|sayMSG| " ")
   (|displayExposedConstructors|)
   (|sayMSG| " ")
   (|displayHiddenConstructors|))
  (dolist (x arg)
    (setq x (|unabbrev| x))
    (when (consp x) (setq x (qcar x)))
```

```
    (cond
     ((null (getdatabase x 'constructorkind))
      (|sayKeyedMsg| 's2iz0049j (list x)))
     ((|member| x (elt |$localExposureData| 2))
      (|sayKeyedMsg| 's2iz0049o (list x |$interpreterFrameName|)))
     (t
      (when (|member| x (elt |$localExposureData| 1))
       (setelt |$localExposureData| 1
        (|delete| x (elt |$localExposureData| 1))))
      (setelt |$localExposureData| 2
        (msort (cons x (elt |$localExposureData| 2))))
      (|clearClams|)
      (|sayKeyedMsg| 's2iz0049q (list x |$interpreterFrameName|)))))))
```

———-

45.19.8 Display exposed groups

[sayKeyedMsg p327]
[centerAndHighlight p**??**]
[$interpreterFrameName p**??**]
[$localExposureData p688]

— **defun displayExposedGroups** —

```
(defun |displayExposedGroups| ()
 "Display exposed groups"
 (declare (special |$interpreterFrameName| |$localExposureData|))
 (|sayKeyedMsg| 's2iz0049a (list |$interpreterFrameName|))
 (if  (null (elt |$localExposureData| 0))
  (|centerAndHighlight| "there are no exposed groups")
  (dolist (c (elt |$localExposureData| 0))
     (|centerAndHighlight| c))))
```

———-

45.19.9 Display exposed constructors

[sayKeyedMsg p327]
[centerAndHighlight p**??**]
[$localExposureData p688]

— **defun displayExposedConstructors** —

```
(defun |displayExposedConstructors| ()
```

```
 "Display exposed constructors"
 (declare (special |$localExposureData|))
 (|sayKeyedMsg| 's2iz0049b nil)
 (if  (null (elt |$localExposureData| 1))
  (|centerAndHighlight| "there are no explicitly exposed constructors")
  (dolist (c (elt |$localExposureData| 1))
     (|centerAndHighlight| c))))
```

———-

45.19.10 Display hidden constructors

[sayKeyedMsg p327]
[centerAndHighlight p??]
[$localExposureData p688]

— **defun displayHiddenConstructors** —

```
(defun |displayHiddenConstructors| ()
 "Display hidden constructors"
 (declare (special |$localExposureData|))
 (|sayKeyedMsg| 's2iz0049c nil)
 (if (null (elt |$localExposureData| 2))
   (|centerAndHighlight| "there are no explicitly hidden constructors")
   (dolist (c (elt |$localExposureData| 2))
     (|centerAndHighlight| c))))
```

———-

45.20 functions

```
            Current Values of  functions  Variables

Variable     Description                            Current Value
-------------------------------------------------------------------
cache        number of function results to cache        0
compile      compile, don't just define function bodies off
recurrence   specially compile recurrence relations     on
```

— **functions** —

```
  (|functions|
```

```
   "some interpreter function options"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{functionscache}
\getchunk{functionscompile}
\getchunk{functionsrecurrence}
  ))
```

———-

45.21 functions cache

```
---------------------- The cache Option -----------------------

 Description: number of function results to cache

 )set functions cache is used to tell AXIOM how many  values
  computed by interpreter functions should be saved.  This can
  save quite a bit of time in recursive functions, though one
  must consider that the cached values will take up (perhaps
  valuable) room in the workspace.

 The value given  after cache must either be the word all or
 a positive integer. This may be followed by any number of
 function names whose cache sizes you wish to so set.  If no
 functions are given, the default cache  size is set.

 Examples:   )set fun cache all
             )set fun cache 10 f g Legendre

 In general, functions will cache no returned values.
```

— **functionscache** —

```
    (|cache|
     "number of function results to cache"
     |interpreter|
     FUNCTION
     |setFunctionsCache|
     NIL
     |htSetCache|)
```

———-

45.22 Variables Used

45.22.1 defvar $cacheAlist

— **initvars** —

```
(defvar |$cacheAlist| nil)
```

———-

45.23 Functions

45.23.1 The top level set functions cache handler

```
\calls{setFunctionsCache}{object2String}
\calls{setFunctionsCache}{describeSetFunctionsCache}
\calls{setFunctionsCache}{sayAllCacheCounts}
\calls{setFunctionsCache}{sayMessage}
\calls{setFunctionsCache}{bright}
\calls{setFunctionsCache}{terminateSystemCommand}
\calls{setFunctionsCache}{countCache}
\usesdollar{setFunctionsCache}{options}
\usesdollar{setFunctionsCache}{cacheCount}
\usesdollar{setFunctionsCache}{cacheAlist}
\begin{chunk}{defun setFunctionsCache}
(defun |setFunctionsCache| (arg)
 "The top level set functions cache handler"
 (let (|$options| n)
  (declare (special |$options| |$cacheCount| |$cacheAlist|))
  (cond
    ((eq arg '|%initialize%|)
      (setq |$cacheCount| 0)
      (setq |$cacheAlist| nil))
    ((eq arg '|%display%|)
      (if (null |$cacheAlist|)
       (|object2String| |$cacheCount|)
       "..."))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
      (|describeSetFunctionsCache|)
      (terpri)
      (|sayAllCacheCounts|))
    (t
      (setq n (car arg))
      (cond
       ((and (not (eq n '|all|)) (or (null (integerp n)) (minusp n)))
```

```
        (|sayMessage|
         `("Your value of" ,@(|bright| n) "is invalid because ..."))
        (|describeSetFunctionsCache|)
        (|terminateSystemCommand|))
       (t
        (when (cdr arg) (list (cons '|vars| (cdr arg))))
        (|countCache| n)))))))
```

```
\end{chunk}

\defunsec{countCache}{Display a particular cache count}
\calls{countCache}{qcdr}
\calls{countCache}{qcar}
\calls{countCache}{identp}
\calls{countCache}{sayKeyedMsg}
\calls{countCache}{insertAlist}
\calls{countCache}{internl}
\calls{countCache}{sayCacheCount}
\calls{countCache}{optionError}
\usesdollar{countCache}{options}
\usesdollar{countCache}{cacheAlist}
\usesdollar{countCache}{cacheCount}
\begin{chunk}{defun countCache}
(defun |countCache| (n)
 "Display a particular cache count"
 (let (tmp1 l cachecountname)
 (declare (special |$options| |$cacheAlist| |$cacheCount|))
 (cond
  (|$options|
   (cond
    ((and (consp |$options|)
          (eq (qcdr |$options|) nil)
          (progn
           (setq tmp1 (qcar |$options|))
           (and (consp tmp1)
                (eq (qcar tmp1) '|vars|)
                (progn (setq l (qcdr tmp1)) t))))
     (dolist (x l)
       (if (null (identp x))
         (|sayKeyedMsg| 's2if0007 (list x))
         (progn
          (setq |$cacheAlist| (|insertAlist| x n |$cacheAlist|))
          (setq cachecountname (internl x ";COUNT"))
          (set cachecountname n)
          (|sayCacheCount| x n)))))
    (t (|optionError| (caar |$options|) nil))))
  (t
   (|sayCacheCount| nil (setq |$cacheCount| n))))))
```

```
\end{chunk}
```

```
\defun{insertAlist}{insertAlist}
\calls{insertAlist}{rplac}
\calls{insertAlist}{?order}
\begin{chunk}{defun insertAlist}
(defun |insertAlist| (a b z)
 (labels (
  (fn (a b z)
   (cond
    ((null (cdr z)) (rplac (cdr z) (list (cons a b))))
    ((equal a (elt (elt z 1) 0)) (rplac (cdr (elt z 1)) b))
    ((?order (elt (elt z 1) 0) a) (rplac (cdr z) (cons (cons a b) (cdr z))))
    (t (fn a b (cdr z))))))
  (cond
    ((null z) (list (cons a b)))
    ((equal a (elt (elt z 0) 0)) (rplac (cdar z) b) z)
    ((?order (elt (elt z 0) 0) a) (cons (cons a b) z))
    (t (fn a b z) z))))

\end{chunk}

\defunsec{describeSetFunctionsCache}{Describe the set functions cache}
\calls{describeSetFunctionsCache}{sayBrightly}
\begin{chunk}{defun describeSetFunctionsCache}
(defun |describeSetFunctionsCache| ()
 "Describe the set functions cache"
 (|sayBrightly| (list
  '|%b| ")set functions cache"
  '|%d| "is used to tell AXIOM how many"
  '|%l| " values computed by interpreter functions should be saved.  This"
  '|%l| " can save quite a bit of time in recursive functions, though one"
  '|%l| " must consider that the cached values will take up (perhaps"
  '|%l| " valuable) room in the workspace."
  '|%l|
  '|%l| " The value given after"
  '|%b| "cache"
  '|%d| "must either be the word"
  '|%b| "all"
  '|%d| "or a positive integer."
  '|%l| " This may be followed by any number of function names whose cache"
  '|%l| " sizes you wish to so set.  If no functions are given, the default"
  '|%l| " cache size is set."
  '|%l|
  '|%l|  " Examples:"
  '|%l|  "   )set fun cache all         )set fun cache 10 f g Legendre")))

\end{chunk}

\defunsec{sayAllCacheCounts}{Display all cache counts}
\calls{sayAllCacheCounts}{sayCacheCount}
```

```
\usesdollar{sayAllCacheCounts}{cacheCount}
\usesdollar{sayAllCacheCounts}{cacheAlist}
\begin{chunk}{defun sayAllCacheCounts}
(defun |sayAllCacheCounts| ()
 "Display all cache counts"
 (let (x n)
 (declare (special |$cacheCount| |$cacheAlist|))
   (|sayCacheCount| nil |$cacheCount|)
   (when |$cacheAlist|
     (do ((t0 |$cacheAlist| (cdr t0)) (t1 nil))
         ((or (atom t0)
              (progn (setq t1 (car t0)) nil)
              (progn
               (progn (setq x (car t1)) (setq n (cdr t1)) t1)
               nil))
             nil)
       (when (not (equal n |$cacheCount|)) (|sayCacheCount| x n))))))

\end{chunk}

\defunsec{sayCacheCount}{Describe the cache counts}
\calls{sayCacheCount}{bright}
\calls{sayCacheCount}{linearFormatName}
\calls{sayCacheCount}{sayBrightly}
\begin{chunk}{defun sayCacheCount}
(defun |sayCacheCount| (fn n)
 "Describe the cache counts"
 (let (prefix phrase)
    (setq prefix
     (cond
      (fn (cons '|function| (|bright| (|linearFormatName| fn))))
      ((eql n 0) (list '|interpreter functions |))
      (t (list '|In general, interpreter functions |))))
    (cond
     ((eql n 0)
      (cond
       (fn
        (|sayBrightly|
         `("   Caching for " ,prefix "is turned off")))
       (t
        (|sayBrightly| " In general, functions will cache no returned values."
         ))))
     (t
      (setq phrase
       (cond
        ((eq n '|all|) `(,@(|bright| '|all|) |values.|))
        ((eql n 1) (list '| only the last value.|))
        (t `(| the last| ,@(|bright| n) |values.|))))
      (|sayBrightly|
       `("   " ,@prefix "will cache" ,@phrase))))))
```

```
\end{chunk}

\section{functions compile}
\begin{verbatim}
---------------------- The compile Option ----------------------

 Description: compile, don't just define function bodies

 The compile option may be followed by any one of the following:

 -> on
    off

 The current setting is indicated.
```

45.23.2 defvar $compileDontDefineFunctions

— **initvars** —

```
(defvar |$compileDontDefineFunctions| t
   "compile, don't just define function bodies")
```

———-

— **functionscompile** —

```
     (|compile|
      "compile, don't just define function bodies"
      |interpreter|
      LITERALS
      |$compileDontDefineFunctions|
      (|on| |off|)
      |on|)
```

———-

45.24 functions recurrence

```
-------------------- The recurrence Option --------------------

 Description: specially compile recurrence relations
```

```
The recurrence option may be followed by any one of the
following:

-> on
   off

The current setting is indicated.
```

45.24.1 defvar $compileRecurrence

— **initvars** —

```
(defvar |$compileRecurrence| t "specially compile recurrence relations")
```

———-

— **functionsrecurrence** —

```
(|recurrence|
 "specially compile recurrence relations"
 |interpreter|
 LITERALS
 |$compileRecurrence|
 (|on| |off|)
 |on|)
```

———-

45.25 fortran

```
             Current Values of  fortran  Variables

Variable     Description                          Current Value
-------------------------------------------------------------
ints2floats  where sensible, coerce integers to reals   on
fortindent   the number of characters indented          6
fortlength   the number of characters on a line         72
typedecs     print type and dimension lines             on
defaulttype  default generic type for FORTRAN object    REAL
precision    precision of generated FORTRAN objects     double
intrinsic    whether to use INTRINSIC FORTRAN functions off
explength    character limit for FORTRAN expressions    1320
segment      split long FORTRAN expressions             on
```

```
optlevel     FORTRAN optimisation level                0
startindex   starting index for FORTRAN arrays         1
calling      options for external FORTRAN calls        ...

Variables with current values of ... have further sub-options.
For example, issue )set calling to see what the options are for
calling.
For more information, issue )help set .
```

— **fortran** —

```
  (|fortran|
   "view and set options for FORTRAN output"
   |interpreter|
   TREE
   |novar|
    (
\getchunk{fortranints2floats}
\getchunk{fortranfortindent}
\getchunk{fortranfortlength}
\getchunk{fortrantypedecs}
\getchunk{fortrandefaulttype}
\getchunk{fortranprecision}
\getchunk{fortranintrinsic}
\getchunk{fortranexplength}
\getchunk{fortransegment}
\getchunk{fortranoptlevel}
\getchunk{fortranstartindex}
\getchunk{fortrancalling}
  ))
```

————-

45.25.1 ints2floats

```
------------------- The ints2floats Option --------------------

 Description: where sensible, coerce integers to reals

 The ints2floats option may be followed by any one of the
 following:

 -> on
    off

 The current setting is indicated.
```

45.25.2 defvar $fortInts2Floats

— **initvars** —

```
(defvar |$fortInts2Floats| t "where sensible, coerce integers to reals")
```

————-

— **fortranints2floats** —

```
(|ints2floats|
 "where sensible, coerce integers to reals"
 |interpreter|
 LITERALS
 |$fortInts2Floats|
 (|on| |off|)
 |on|)
```

————-

45.25.3 fortindent

```
-------------------- The fortindent Option --------------------

 Description: the number of characters indented

 The fortindent option may be followed by an integer in the range
 0 to inclusive. The current setting is 6
```

45.25.4 defvar $fortIndent

— **initvars** —

```
(defvar |$fortIndent| 6 "the number of characters indented")
```

————-

— **fortranfortindent** —

```
(|fortindent|
 "the number of characters indented"
```

```
      |interpreter|
      INTEGER
      |$fortIndent|
      (0 NIL)
      6)
```

———-

45.25.5 fortlength

```
-------------------- The fortlength Option --------------------

 Description: the number of characters on a line

 The fortlength option may be followed by an integer in the range
 1 to inclusive. The current setting is 72
```

45.25.6 defvar $fortLength

— **initvars** —

```
(defvar |$fortLength| 72 "the number of characters on a line")
```

———-

— **fortranfortlength** —

```
     (|fortlength|
      "the number of characters on a line"
      |interpreter|
      INTEGER
      |$fortLength|
      (1 NIL)
      72)
```

———-

45.25.7 typedecs

```
--------------------- The typedecs Option ---------------------

 Description: print type and dimension lines
```

```
The typedecs option may be followed by any one of the
following:

-> on
   off

The current setting is indicated.
```

45.25.8 defvar $printFortranDecs

— **initvars** —

```
(defvar |$printFortranDecs| t "print type and dimension lines")
```

———-

— **fortrantypedecs** —

```
    (|typedecs|
     "print type and dimension lines"
     |interpreter|
     LITERALS
     |$printFortranDecs|
     (|on| |off|)
     |on|)
```

———-

45.25.9 defaulttype

```
------------------- The defaulttype Option --------------------

 Description: default generic type for FORTRAN object

 The defaulttype option may be followed by any one of the
 following:

 -> REAL
    INTEGER
    COMPLEX
    LOGICAL
    CHARACTER

 The current setting is indicated.
```

45.25.10 defvar $defaultFortranType

— **initvars** —

```
(defvar |$defaultFortranType| 'real "default generic type for FORTRAN object")
```

————-

— **fortrandefaulttype** —

```
    (|defaulttype|
     "default generic type for FORTRAN object"
     |interpreter|
     LITERALS
     |$defaultFortranType|
     (REAL INTEGER COMPLEX LOGICAL CHARACTER)
     REAL)
```

————-

45.25.11 precision

```
-------------------- The precision Option ---------------------

 Description: precision of generated FORTRAN objects

 The precision option may be followed by any one of the
 following:

    single
 -> double

 The current setting is indicated.
```

45.25.12 defvar $fortranPrecision

— **initvars** —

```
(defvar |$fortranPrecision| '|double| "precision of generated FORTRAN objects")
```

————-

— **fortranprecision** —

```
(|precision|
"precision of generated FORTRAN objects"
 |interpreter|
 LITERALS
 |$fortranPrecision|
 (|single| |double|)
 |double|)
```

———-

45.25.13 intrinsic

```
-------------------- The intrinsic Option ---------------------

 Description: whether to use INTRINSIC FORTRAN functions

 The intrinsic option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.25.14 defvar $useIntrinsicFunctions

— **initvars** —

```
(defvar |$useIntrinsicFunctions| nil
   "whether to use INTRINSIC FORTRAN functions")
```

———-

— **fortranintrinsic** —

```
(|intrinsic|
 "whether to use INTRINSIC FORTRAN functions"
 |interpreter|
 LITERALS
 |$useIntrinsicFunctions|
 (|on| |off|)
 |off|)
```

———-

45.25.15 explength

```
-------------------- The explength Option ---------------------

 Description: character limit for FORTRAN expressions

 The explength option may be followed by an integer in the range
 0 to inclusive. The current setting is 1320
```

45.25.16 defvar $maximumFortranExpressionLength

— **initvars** —

```
(defvar |$maximumFortranExpressionLength| 1320
  "character limit for FORTRAN expressions")
```

———-

— **fortranexplength** —

```
     (|explength|
      "character limit for FORTRAN expressions"
      |interpreter|
      INTEGER
      |$maximumFortranExpressionLength|
      (0 NIL)
      1320)
```

———-

45.25.17 segment

```
---------------------- The segment Option ----------------------

 Description: split long FORTRAN expressions

 The segment option may be followed by any one of the following:

 -> on
    off
```

```
 The current setting is indicated.
```

45.25.18 defvar $fortranSegment

— **initvars** —

```
(defvar |$fortranSegment| t "split long FORTRAN expressions")
```

————-

— **fortransegment** —

```
      (|segment|
       "split long FORTRAN expressions"
       |interpreter|
       LITERALS
       |$fortranSegment|
       (|on| |off|)
       |on|)
```

————-

45.25.19 optlevel

```
---------------------- The optlevel Option ----------------------

 Description: FORTRAN optimisation level

 The optlevel option may be followed by an integer in the range
 0 to 2 inclusive. The current setting is 0
```

45.25.20 defvar $fortranOptimizationLevel

— **initvars** —

```
(defvar |$fortranOptimizationLevel| 0 "FORTRAN optimisation level")
```

————-

— **fortranoptlevel** —

```
(|optlevel|
 "FORTRAN optimisation level"
 |interpreter|
 INTEGER
 |$fortranOptimizationLevel|
 (0 2)
 0)
```

———-

45.25.21 startindex

```
-------------------- The startindex Option --------------------

 Description: starting index for FORTRAN arrays

 The startindex option may be followed by an integer in the range
 0 to 1 inclusive. The current setting is 1
```

45.25.22 defvar $fortranArrayStartingIndex

— **initvars** —

```
(defvar |$fortranArrayStartingIndex| 1 "starting index for FORTRAN arrays")
```

———-

— **fortranstartindex** —

```
(|startindex|
 "starting index for FORTRAN arrays"
 |interpreter|
 INTEGER
 |$fortranArrayStartingIndex|
 (0 1)
 1)
```

———-

45.25.23 calling

```
Current Values of  calling  Variables
```

```
Variable     Description                              Current Value
-------------------------------------------------------------------
tempfile     set location of temporary data files       /tmp/
directory    set location of generated FORTRAN files    ./
linker       linker arguments (e.g. libraries to search) -lxlf
```

— **fortrancalling** —

```
      (|calling|
      "options for external FORTRAN calls"
      |interpreter|
      TREE
      |novar|
      (
\getchunk{callingtempfile}
\getchunk{callingdirectory}
\getchunk{callinglinker}
       )
      )
```

———-

tempfile

```
---------------------- The tempfile Option ----------------------

 Description: set location of temporary data files

 )set fortran calling tempfile  is used to tell AXIOM where
 to place intermediate FORTRAN data files . This must be the
 name of a valid existing directory to which you have permission
 to write (including the final slash).

 Syntax:
   )set fortran calling tempfile DIRECTORYNAME

 The current setting is /tmp/
```

45.25.24 defvar $fortranTmpDir

— **initvars** —

```
(defvar |$fortranTmpDir| "/tmp/" "set location of temporary data files")
```

———-

— **callingtempfile** —

```
(|tempfile|
 "set location of temporary data files"
 |interpreter|
 FUNCTION
 |setFortTmpDir|
 (("enter directory name for which you have write-permission"
   DIRECTORY
   |$fortranTmpDir|
   |chkDirectory|
   "/tmp/"))
 NIL)
```

———-

45.25.25 The top level set fortran calling tempfile handler

[pname p1039]
[describeSetFortTmpDir p716]
[validateOutputDirectory p716]
[sayBrightly p**??**]
[bright p**??**]
[$fortranTmpDir p714]

— **defun setFortTmpDir** —

```
(defun |setFortTmpDir| (arg)
 "The top level set fortran calling tempfile handler"
 (let (mode)
   (declare (special |$fortranTmpDir|))
   (cond
    ((eq arg '|%initialize%|) (setq |$fortranTmpDir| "/tmp/"))
    ((eq arg '|%display%|)
     (if (stringp |$fortranTmpDir|)
      |$fortranTmpDir|
      (pname |$fortranTmpDir|)))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetFortTmpDir|))
    ((null (setq mode (|validateOutputDirectory| arg)))
     (|sayBrightly|
      `(" Sorry, but your argument(s)" ,@(|bright| arg)
            "is(are) not valid." |%l|))
      (|describeSetFortTmpDir|))
    (t (setq |$fortranTmpDir| mode)))))
```

———-

45.25.26 Validate the output directory

— defun validateOutputDirectory —

```
(defun |validateOutputDirectory| (x)
 "Validate the output directory"
 (let ((dirname (car x)))
  (when (and (pathname-directory dirname) (null (probe-file dirname)))
   dirname)))
```

———-

45.25.27 Describe the set fortran calling tempfile

[sayBrightly p??]
[$fortranTmpDir p714]

— defun describeSetFortTmpDir —

```
(defun |describeSetFortTmpDir| ()
 "Describe the set fortran calling tempfile"
 (declare (special |$fortranTmpDir|))
 (|sayBrightly| (list
  '|%b| ")set fortran calling tempfile"
  '|%d| " is used to tell AXIOM where"
  '|%l| " to place intermediate FORTRAN data files . This must be the "
  '|%l| " name of a valid existing directory to which you have permission "
  '|%l| " to write (including the final slash)."
  '|%l|
  '|%l| " Syntax:"
  '|%l| "   )set fortran calling tempfile DIRECTORYNAME"
  '|%l|
  '|%l| " The current setting is"
  '|%b| |$fortranTmpDir|
  '|%d|)))
```

———-

directory

```
-------------------- The directory Option ---------------------
```

```
Description: set location of generated FORTRAN files

)set fortran calling directory  is used to tell AXIOM where
to place generated FORTRAN files. This must be the name
of a valid existing directory to which you have permission
to write (including the final slash).

Syntax:
  )set fortran calling directory DIRECTORYNAME

The current setting is ./
```

45.25.28 defvar $fortranDirectory

— **initvars** —

```
(defvar |$fortranDirectory| "./" "set location of generated FORTRAN files")
```

———-

— **callingdirectory** —

```
(|directory|
 "set location of generated FORTRAN files"
 |interpreter|
 FUNCTION
 |setFortDir|
 (("enter directory name for which you have write-permission"
   DIRECTORY
   |$fortranDirectory|
   |chkDirectory|
   "./"))
 NIL)
```

———-

45.25.29 defun setFortDir

[pname p1039]
[describeSetFortDir p718]
[validateOutputDirectory p716]
[sayBrightly p??]
[bright p??]

[$fortranDirectory p717]

— **defun setFortDir** —

```
(defun |setFortDir| (arg)
 (declare (special |$fortranDirectory|))
 (let (mode)
   (COND
    ((eq arg '|%initialize%|) (setq |$fortranDirectory| "./"))
    ((eq arg '|%display%|)
     (if (stringp |$fortranDirectory|)
      |$fortranDirectory|
      (pname |$fortranDirectory|)))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
      (|describeSetFortDir|))
    ((null (setq mode (|validateOutputDirectory| arg)))
     (|sayBrightly|
      `(" Sorry, but your argument(s)" ,@(|bright| arg)
        "is(are) not valid." |%l|))
     (|describeSetFortDir|))
    (t (setq |$fortranDirectory| mode)))))
```

———-

45.25.30 defun describeSetFortDir

[sayBrightly p??]
[$fortranDirectory p717]

— **defun describeSetFortDir** —

```
(defun |describeSetFortDir| ()
 (declare (special |$fortranDirectory|))
 (|sayBrightly| (list
  '|%b| ")set fortran calling directory"
  '|%d| " is used to tell AXIOM where"
  '|%l| " to place generated FORTRAN files. This must be the name "
  '|%l| " of a valid existing directory to which you have permission "
  '|%l| " to write (including the final slash)."
  '|%l|
  '|%l| " Syntax:"
  '|%l| "   )set fortran calling directory DIRECTORYNAME"
  '|%l|
  '|%l| " The current setting is"
  '|%b| |$fortranDirectory|
  '|%d|)))
```

———-

linker

```
---------------------- The linker Option ----------------------

 Description: linker arguments (e.g. libraries to search)

 )set fortran calling linkerargs is used to pass arguments to
 the linker when using  mkFort  to create functions which call
 Fortran code. For example, it might give a list of libraries
 to be searched, and their locations.
 The string is passed verbatim, so must be the correct syntax for
 the particular linker being used.

 Example: )set fortran calling linker "-lxlf"

 The current setting is -lxlf
```

45.25.31 defvar $fortranLibraries

— **initvars** —

```
(defvar |$fortranLibraries| "-lxlf"
  "linker arguments (e.g. libraries to search)")
```

———-

— **callinglinker** —

```
        (|linker|
         "linker arguments (e.g. libraries to search)"
         |interpreter|
         FUNCTION
         |setLinkerArgs|
         (("enter linker arguments "
           STRING
           |$fortranLibraries|
           |chkDirectory|
           "-lxlf"))
         NIL
         )
```

———-

45.25.32 defun setLinkerArgs

[object2String p??]
[describeSetLinkerArgs p720]
[$fortranLibraries p719]

— **defun setLinkerArgs** —

```
(defun |setLinkerArgs| (arg)
 (declare (special |$fortranLibraries|))
 (cond
  ((eq arg '|%initialize%|) (setq |$fortranLibraries| "-lxlf"))
  ((eq arg '|%display%|) (|object2String| |$fortranLibraries|))
  ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
    (|describeSetLinkerArgs|))
  ((and (listp arg) (stringp (car arg)))
    (setq |$fortranLibraries| (car arg)))
  (t (|describeSetLinkerArgs|))))
```

———-

45.25.33 defun describeSetLinkerArgs

[sayBrightly p??]
[$fortranLibraries p719]

— **defun describeSetLinkerArgs** —

```
(defun |describeSetLinkerArgs| ()
 (declare (special |$fortranLibraries|))
 (|sayBrightly| (list
  '|%b| ")set fortran calling linkerargs"
  '|%d| " is used to pass arguments to the linker"
  '|%l| " when using "
  '|%b| "mkFort"
  '|%d| " to create functions which call Fortran code."
  '|%l| " For example, it might give a list of libraries to be searched,"
  '|%l| " and their locations."
  '|%l| " The string is passed verbatim, so must be the correct syntax for"
  '|%l| " the particular linker being used."
  '|%l|
  '|%l| " Example: )set fortran calling linker \"-lxlf\""
  '|%l|
  '|%l| " The current setting is"
  '|%b| |$fortranLibraries|
  '|%d|)))
```

———-

45.26 hyperdoc

```
            Current Values of  hyperdoc  Variables

Variable     Description                            Current Value
-----------------------------------------------------------------
fullscreen   use full screen for this facility          off
mathwidth    screen width for history output            120
```

— **hyperdoc** —

```
  (|hyperdoc|
   "options in using HyperDoc"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{hyperdocfullscreen}
\getchunk{hyperdocmathwidth}
   ))
```

———-

45.26.1 fullscreen

```
-------------------- The fullscreen Option --------------------

 Description: use full screen for this facility

 The fullscreen option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.26.2 defvar $fullScreenSysVars

— **initvars** —

```
(defvar |$fullScreenSysVars| nil "use full screen for this facility")
```

———-

— **hyperdocfullscreen** —

```
(|fullscreen|
 "use full screen for this facility"
 |interpreter|
 LITERALS
 |$fullScreenSysVars|
 (|on| |off|)
 |off|)
```

———-

45.26.3 mathwidth

```
-------------------- The mathwidth Option ---------------------

 Description: screen width for history output

 The mathwidth option may be followed by an integer in the range
 0 to inclusive. The current setting is 120
```

45.26.4 defvar $historyDisplayWidth

— **initvars** —

```
(defvar |$historyDisplayWidth| 120 "screen width for history output")
```

———-

— **hyperdocmathwidth** —

```
(|mathwidth|
 "screen width for history output"
 |interpreter|
 INTEGER
 |$historyDisplayWidth|
 (0 NIL)
 120)
```

———-

45.27 help

```
              Current Values of  help  Variables

Variable     Description                              Current Value
-----------------------------------------------------------------
fullscreen   use fullscreen facility, if possible       on
```

— **help** —

```
  (|help|
   "view and set some help options"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{helpfullscreen}
   ))
```

———-

45.27.1 fullscreen

```
-------------------- The fullscreen Option --------------------

 Description: use fullscreen facility, if possible

 The fullscreen option may be followed by any one of the
 following:

 -> on
   off

 The current setting is indicated.
```

45.27.2 defvar $useFullScreenHelp

— **initvars** —

```
(defvar |$useFullScreenHelp| t "use fullscreen facility, if possible")
```

———-

— **helpfullscreen** —

```
(|fullscreen|
 "use fullscreen facility, if possible"
 |interpreter|
 LITERALS
 |$useFullScreenHelp|
 (|on| |off|)
 |on|)
```

———-

45.28 history

```
---------------------- The history Option ----------------------

 Description: save workspace values in a history file

 The history option may be followed by any one of the
 following:

 -> on
    off

 The current setting is indicated.
```

45.28.1 defvar $HiFiAccess

— **initvars** —

```
(defvar |$HiFiAccess| t "save workspace values in a history file")
```

———-

— **history** —

```
(|history|
 "save workspace values in a history file"
 |interpreter|
 LITERALS
```

```
|$HiFiAccess|
(|on| |off|)
|on|)
```

———-

45.29 messages

```
            Current Values of  messages  Variables

Variable     Description                              Current Value
-----------------------------------------------------------------
autoload     print file auto-load messages              off
bottomup     display bottom up modemap selection        off
coercion     display datatype coercion messages         off
dropmap      display old map defn when replaced         off
expose       warning for unexposed functions            off
file         print msgs also to SPADMSG LISTING         off
frame        display messages about frames              off
highlighting use highlighting in system messages        off
instant      present instantiation summary              off
insteach     present instantiation info                 off
interponly   say when function code is interpreted      on
number       display message number with message        off
prompt       set type of input prompt to display        step
selection    display function selection msgs            off
set          show )set setting after assignment         off
startup      display messages on start-up               off
summary      print statistics after computation         off
testing      print system testing header                off
time         print timings after computation            off
type         print type after computation               on
void         print Void value when it occurs            off
any          print the internal type of objects of domain Any on
naglink      show NAGLink messages                      on
```

— **messages** —

```
  (|messages|
   "show messages for various system features"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{messagesany}
\getchunk{messagesautoload}
```

```
\getchunk{messagesbottomup}
\getchunk{messagescoercion}
\getchunk{messagesdropmap}
\getchunk{messagesexpose}
\getchunk{messagesfile}
\getchunk{messagesframe}
\getchunk{messageshighlighting}
\getchunk{messagesinstant}
\getchunk{messagesinsteach}
\getchunk{messagesinterponly}
\getchunk{messagesnaglink}
\getchunk{messagesnumber}
\getchunk{messagesprompt}
\getchunk{messagesselection}
\getchunk{messagesset}
\getchunk{messagesstartup}
\getchunk{messagessummary}
\getchunk{messagestesting}
\getchunk{messagestime}
\getchunk{messagestype}
\getchunk{messagesvoid}
   ))
```

———-

45.29.1 any

```
----------------------- The any Option ------------------------

 Description: print the internal type of objects of domain Any

 The any option may be followed by any one of the following:

 -> on
    off

 The current setting is indicated.
```

45.29.2 defvar $printAnyIfTrue

— **initvars** —

```
(defvar |$printAnyIfTrue| t
   "print the internal type of objects of domain Any")
```

———-

— **messagesany** —

```
(|any|
 "print the internal type of objects of domain Any"
 |interpreter|
 LITERALS
 |$printAnyIfTrue|
 (|on| |off|)
 |on|)
```

———-

45.29.3 autoload

```
---------------------- The autoload Option ----------------------

 Description: print file auto-load messages
```

45.29.4 defvar $printLoadMsgs

— **initvars** —

```
(defvar |$printLoadMsgs| nil "print file auto-load messages")
```

———-

— **messagesautoload** —

```
(|autoload|
 "print file auto-load messages"
 |interpreter|
 LITERALS
 |$printLoadMsgs|
 (|on| |off|)
 |on|)
```

———-

45.29.5 bottomup

```
--------------------- The bottomup Option ---------------------

 Description: display bottom up modemap selection

 The bottomup option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.29.6 defvar $reportBottomUpFlag

— **initvars** —

```
(defvar |$reportBottomUpFlag| nil "display bottom up modemap selection")
```

————-

— **messagesbottomup** —

```
     (|bottomup|
      "display bottom up modemap selection"
      |development|
      LITERALS
      |$reportBottomUpFlag|
      (|on| |off|)
      |off|)
```

————-

45.29.7 coercion

```
--------------------- The coercion Option ---------------------

 Description: display datatype coercion messages

 The coercion option may be followed by any one of the
 following:
```

```
   on
-> off

The current setting is indicated.
```

45.29.8 defvar $reportCoerceIfTrue

— **initvars** —

```
(defvar |$reportCoerceIfTrue| nil "display datatype coercion messages")
```

———-

— **messagescoercion** —

```
    (|coercion|
     "display datatype coercion messages"
     |development|
     LITERALS
     |$reportCoerceIfTrue|
     (|on| |off|)
     |off|)
```

———-

45.29.9 dropmap

```
---------------------- The dropmap Option -----------------------

 Description: display old map defn when replaced

 The dropmap option may be followed by any one of the
 following:

   on
 -> off

 The current setting is indicated.
```

45.29.10 defvar $displayDroppedMap

— **initvars** —

```
(defvar |$displayDroppedMap| nil "display old map defn when replaced")
```

————-

— **messagesdropmap** —

```
    (|dropmap|
     "display old map defn when replaced"
     |interpreter|
     LITERALS
     |$displayDroppedMap|
     (|on| |off|)
     |off|)
```

————-

45.29.11 expose

```
---------------------- The expose Option ----------------------

 Description: warning for unexposed functions

 The expose option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.29.12 defvar $giveExposureWarning

— **initvars** —

```
(defvar |$giveExposureWarning| nil "warning for unexposed functions")
```

————-

— **messagesexpose** —

```
(|expose|
 "warning for unexposed functions"
 |interpreter|
 LITERALS
 |$giveExposureWarning|
 (|on| |off|)
 |off|)
```

———-

45.29.13 file

```
----------------------- The file Option -----------------------

 Description: print msgs also to SPADMSG LISTING

 The file option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.29.14 defvar $printMsgsToFile

— **initvars** —

```
(defvar |$printMsgsToFile| nil "print msgs also to SPADMSG LISTING")
```

———-

— **messagesfile** —

```
(|file|
 "print msgs also to SPADMSG LISTING"
 |development|
 LITERALS
 |$printMsgsToFile|
 (|on| |off|)
```

```
      |off|)
```

———-

45.29.15 frame

```
---------------------- The frame Option -----------------------

 Description: display messages about frames

 The frame option may be followed by any one of the following:

    on
 -> off

 The current setting is indicated.
```

45.29.16 defvar $frameMessages

— **initvars** —

```
(defvar |$frameMessages| nil "display messages about frames")
```

———-

— **messagesframe** —

```
     (|frame|
      "display messages about frames"
      |interpreter|
      LITERALS
      |$frameMessages|
      (|on| |off|)
      |off|)
```

———-

45.29.17 highlighting

```
------------------- The highlighting Option -------------------
```

```
Description: use highlighting in system messages

The highlighting option may be followed by any one of the
following:

   on
-> off

The current setting is indicated.
```

45.29.18 defvar $highlightAllowed

— **initvars** —

```
(defvar |$highlightAllowed| nil "use highlighting in system messages")
```

————-

— **messageshighlighting** —

```
    (|highlighting|
     "use highlighting in system messages"
     |interpreter|
     LITERALS
     |$highlightAllowed|
     (|on| |off|)
     |off|)
```

————-

45.29.19 instant

```
---------------------- The instant Option -----------------------

 Description: present instantiation summary

 The instant option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.29.20 defvar $reportInstantiations

— **initvars** —

```
(defvar |$reportInstantiations| nil "present instantiation summary")
```

———-

— **messagesinstant** —

```
    (|instant|
     "present instantiation summary"
     |development|
     LITERALS
     |$reportInstantiations|
     (|on| |off|)
     |off|)
```

———-

45.29.21 insteach

```
---------------------- The insteach Option ---------------------

 Description: present instantiation info

 The insteach option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.29.22 defvar $reportEachInstantiation—

— **initvars** —

```
(defvar |$reportEachInstantiation| nil "present instantiation info")
```

———-

— **messagesinsteach** —

```
(|insteach|
 "present instantiation info"
 |development|
 LITERALS
 |$reportEachInstantiation|
 (|on| |off|)
 |off|)
```

————-

45.29.23 interponly

```
-------------------- The interponly Option --------------------

 Description: say when function code is interpreted

 The interponly option may be followed by any one of the
 following:

 -> on
    off

 The current setting is indicated.
```

45.29.24 defvar $reportInterpOnly

— **initvars** —

```
(defvar |$reportInterpOnly| t "say when function code is interpreted")
```

————-

— **messagesinterponly** —

```
(|interponly|
 "say when function code is interpreted"
 |interpreter|
 LITERALS
 |$reportInterpOnly|
 (|on| |off|)
```

```
      |on|)
```

———-

45.29.25 naglink

```
---------------------- The naglink Option ----------------------

 Description: show NAGLink messages

 The naglink option may be followed by any one of the
 following:

 -> on
    off

 The current setting is indicated.
```

45.29.26 defvar $nagMessages

— **initvars** —

```
(defvar |$nagMessages| t "show NAGLink messages")
```

———-

— **messagesnaglink** —

```
     (|naglink|
      "show NAGLink messages"
      |interpreter|
      LITERALS
      |$nagMessages|
      (|on| |off|)
      |on|)
```

———-

45.29.27 number

```
---------------------- The number Option ----------------------
```

```
 Description: display message number with message

 The number option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.29.28 defvar $displayMsgNumber

— **initvars** —

```
(defvar |$displayMsgNumber| nil "display message number with message")
```

———-

— **messagesnumber** —

```
     (|number|
      "display message number with message"
      |interpreter|
      LITERALS
      |$displayMsgNumber|
      (|on| |off|)
      |off|)
```

———-

45.29.29 prompt

```
---------------------- The prompt Option ----------------------

 Description: set type of input prompt to display

 The prompt option may be followed by any one of the following:

    none
    frame
    plain
 -> step
```

```
   verbose

The current setting is indicated.
```

45.29.30 defvar $inputPromptType

— **initvars** —

```
(defvar |$inputPromptType| '|step| "set type of input prompt to display")
```

———-

— **messagesprompt** —

```
(|prompt|
 "set type of input prompt to display"
 |interpreter|
 LITERALS
 |$inputPromptType|
 (|none| |frame| |plain| |step| |verbose|)
 |step|)
```

———-

45.29.31 selection

```
-------------------- The selection Option ---------------------

Description: display function selection msgs

The selection option may be followed by any one of the
following:

   on
-> off

The current setting is indicated.
```

TPDHERE: This is a duplicate of)set mes bot on because both use the `$reportBottomUpFlag` flag

— **messagesselection** —

```
     (|selection|
      "display function selection msgs"
      |interpreter|
      LITERALS
      |$reportBottomUpFlag|
      (|on| |off|)
      |off|)
```

———-

45.29.32 set

```
---------------------- The set Option ------------------------

 Description: show )set setting after assignment

 The set option may be followed by any one of the following:

    on
 -> off

 The current setting is indicated.
```

45.29.33 defvar $displaySetValue

— **initvars** —

```
(defvar |$displaySetValue| nil "show )set setting after assignment")
```

———-

— **messagesset** —

```
     (|set|
      "show )set setting after assignment"
      |interpreter|
      LITERALS
      |$displaySetValue|
      (|on| |off|)
      |off|)
```

———-

45.29.34 startup

```
---------------------- The startup Option ----------------------

 Description: display messages on start-up

 The startup option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.29.35 defvar $displayStartMsgs

— **initvars** —

```
(defvar |$displayStartMsgs| t "display messages on start-up")
```

————-

— **messagesstartup** —

```
     (|startup|
      "display messages on start-up"
      |interpreter|
      LITERALS
      |$displayStartMsgs|
      (|on| |off|)
      |on|)
```

————-

45.29.36 summary

```
---------------------- The summary Option ----------------------

 Description: print statistics after computation

 The summary option may be followed by any one of the
 following:
```

```
   on
-> off

The current setting is indicated.
```

45.29.37 defvar $printStatisticsSummaryIfTrue

— **initvars** —

```
(defvar |$printStatisticsSummaryIfTrue| nil
     "print statistics after computation")
```

———-

— **messagessummary** —

```
    (|summary|
     "print statistics after computation"
     |interpreter|
     LITERALS
     |$printStatisticsSummaryIfTrue|
     (|on| |off|)
     |off|)
```

———-

45.29.38 testing

```
---------------------- The testing Option -----------------------

 Description: print system testing header

 The testing option may be followed by any one of the
 following:

   on
 -> off

 The current setting is indicated.
```

45.29.39 defvar $testingSystem

— **initvars** —

```
(defvar |$testingSystem| nil "print system testing header")
```

———-

— **messagestesting** —

```
     (|testing|
      "print system testing header"
      |development|
      LITERALS
      |$testingSystem|
      (|on| |off|)
      |off|)
```

———-

45.29.40 time

```
---------------------- The time Option ----------------------

 Description: print timings after computation

 The time option may be followed by any one of the following:

    on
 -> off
    long

 The current setting is indicated.
```

45.29.41 defvar $printTimeIfTrue

— **initvars** —

```
(defvar |$printTimeIfTrue| nil "print timings after computation")
```

———-

— **messagestime** —

```
(|time|
 "print timings after computation"
 |interpreter|
 LITERALS
 |$printTimeIfTrue|
 (|on| |off| |long|)
 |off|)
```

———-

45.29.42 type

```
---------------------- The type Option -----------------------

 Description: print type after computation

 The type option may be followed by any one of the following:

 -> on
    off

 The current setting is indicated.
```

45.29.43 defvar $printTypeIfTrue

— **initvars** —

```
(defvar |$printTypeIfTrue| t "print type after computation")
```

———-

— **messagestype** —

```
(|type|
 "print type after computation"
 |interpreter|
 LITERALS
 |$printTypeIfTrue|
 (|on| |off|)
 |on|)
```

———-

45.29.44 void

```
---------------------- The void Option ----------------------

 Description: print Void value when it occurs

 The void option may be followed by any one of the following:

    on
 -> off

 The current setting is indicated.
```

45.29.45 defvar $printVoidIfTrue

— **initvars** —

```
(defvar |$printVoidIfTrue| nil "print Void value when it occurs")
```

———-

— **messagesvoid** —

```
(|void|
 "print Void value when it occurs"
 |interpreter|
 LITERALS
 |$printVoidIfTrue|
 (|on| |off|)
 |off|)
```

———-

45.30 naglink

```
              Current Values of  naglink  Variables

Variable     Description                          Current Value
-------------------------------------------------------------------
```

```
host         internet address of host for NAGLink       localhost
persistence  number of (fortran) functions to remember  1
messages     show NAGLink messages                      on
double       enforce DOUBLE PRECISION ASPs              on
```

— **naglink** —

```
  (|naglink|
   "options for NAGLink"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{naglinkhost}
\getchunk{naglinkpersistence}
\getchunk{naglinkmessages}
\getchunk{naglinkdouble}
   ))
```

————-

45.30.1 host

```
----------------------- The host Option -----------------------

 Description: internet address of host for NAGLink

 )set naglink host is used to tell  AXIOM which  host to contact
 for a NAGLink request. An Internet address should be supplied.
 The host specified must be running the NAGLink daemon.

 The current setting is localhost
```

45.30.2 defvar $nagHost

— **initvars** —

```
(defvar |$nagHost| "localhost" "internet address of host for NAGLink")
```

————-

— **naglinkhost** —

```
     (|host|
      "internet address of host for NAGLink"
      |interpreter|
      FUNCTION
      |setNagHost|
      (("enter host name"
        DIRECTORY
        |$nagHost|
        |chkDirectory|
        "localhost"))
      NIL)
```

———-

45.30.3 defun setNagHost

[object2String p**??**]
[describeSetNagHost p746]
[$nagHost p745]

— **defun setNagHost** —

```
(defun |setNagHost| (arg)
 (declare (special |$nagHost|))
 (cond
  ((eq arg '|%initialize%|) (setq |$nagHost| "localhost"))
  ((eq arg '|%display%|) (|object2String| |$nagHost|))
  ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
   (|describeSetNagHost|))
  (t (setq |$nagHost| (|object2String| arg)))))
```

———-

45.30.4 defun describeSetNagHost

[sayBrightly p**??**]
[$nagHost p745]

— **defun describeSetNagHost** —

```
(defun |describeSetNagHost| ()
 (declare (special |$nagHost|))
 (|sayBrightly| (list
  '|%b| ")set naglink host"
  '|%d| "is used to tell  AXIOM which  host to contact for"
```

```
  '|%l| " a NAGLink request. An Internet address should be supplied. The host"
  '|%l| " specified must be running the NAGLink daemon."
  '|%l|
  '|%l| " The current setting is"
  '|%b| |$nagHost|
  '|%d|)))
```

———-

45.30.5 persistence

```
------------------- The persistence Option --------------------

 Description: number of (fortran) functions to remember

 )set naglink persistence is used to tell  the  nagd  daemon how
  many ASP source and object files to keep around in case you
  reuse them. This helps to avoid needless recompilations. The
  number specified should be a non-negative integer.

 The current setting is 1
```

45.30.6 defvar $fortPersistence

— **initvars** —

```
(defvar |$fortPersistence| 1 "number of (fortran) functions to remember")
```

———-

— **naglinkpersistence** —

```
    (|persistence|
     "number of (fortran) functions to remember"
     |interpreter|
     FUNCTION
     |setFortPers|
     (("Requested remote storage (for asps):"
      INTEGER
      |$fortPersistence|
      (0 NIL)
      10))
     NIL)
```

———-

45.30.7 defun setFortPers

[describeFortPersistence p748]
[sayMessage p**??**]
[bright p**??**]
[terminateSystemCommand p448]
[$fortPersistence p747]

— defun setFortPers —

```
(defun |setFortPers| (arg)
 (let (n)
 (declare (special |$fortPersistence|))
   (cond
    ((eq arg '|%initialize%|) (setq |$fortPersistence| 1))
    ((eq arg '|%display%|) |$fortPersistence|)
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeFortPersistence|))
    (t
     (setq n (car arg))
     (cond
      ((or (null (integerp n)) (minusp n))
       (|sayMessage|
        `("Your value of" ,@(|bright| n) "is invalid because ..."))
        (|describeFortPersistence|)
        (|terminateSystemCommand|))
      (t (setq |$fortPersistence| (car arg))))))))
```

———-

45.30.8 defun describeFortPersistence

[sayBrightly p**??**]
[$fortPersistence p747]

— defun describeFortPersistence —

```
(defun |describeFortPersistence| ()
 (declare (special |$fortPersistence|))
 (|sayBrightly| (list
 '|%b| ")set naglink persistence"
 '|%d| "is used to tell  the "
 '|%b| '|nagd|
 '|%d| '| daemon how  many ASP|
```

```
 '|%l|
 " source and object files to keep around in case you reuse them. This helps"
 '|%l| " to avoid needless recompilations. The number specified should be a "
 '|%l| " non-negative integer."
 '|%l|
 '|%l| " The current setting is"
 '|%b| |$fortPersistence|
 '|%d|)))
```

———-

45.30.9 messages

```
---------------------- The messages Option ----------------------

 Description: show NAGLink messages

 The messages option may be followed by any one of the
 following:

 -> on
    off

 The current setting is indicated.
```

TPDHERE: this is the same as)set nag mes on

— **naglinkmessages** —

```
    (|messages|
     "show NAGLink messages"
     |interpreter|
     LITERALS
     |$nagMessages|
     (|on| |off|)
      |on|)
```

———-

45.30.10 double

```
---------------------- The double Option ----------------------

 Description: enforce DOUBLE PRECISION ASPs
```

```
 The double option may be followed by any one of the following:

 -> on
    off

 The current setting is indicated.
```

45.30.11 defvar $nagEnforceDouble

— **initvars** —

```
(defvar |$nagEnforceDouble| t "enforce DOUBLE PRECISION ASPs")
```

————-

— **naglinkdouble** —

```
     (|double|
      "enforce DOUBLE PRECISION ASPs"
      |interpreter|
      LITERALS
      |$nagEnforceDouble|
      (|on| |off|)
      |on|)
```

————-

45.31 output

The result of the **)set output** command is:

```
Variable     Description                              Current Value
------------------------------------------------------------------
abbreviate   abbreviate type names                      off
algebra      display output in algebraic form           On:CONSOLE
characters   choose special output character set        plain
fortran      create output in FORTRAN format            Off:CONSOLE
fraction     how fractions are formatted                vertical
html         create output in HTML style                Off:CONSOLE
length       line length of output displays             77
mathml       create output in MathML style              Off:CONSOLE
openmath     create output in OpenMath style            Off:CONSOLE
```

```
script      display output in SCRIPT formula format  Off:CONSOLE
scripts     show subscripts,... linearly             off
showeditor  view output of )show in editor           off
tex         create output in TeX style               Off:CONSOLE
```

Since the output option has a bunch of sub-options each suboption is defined within the output structure.

— **output** —

```
  (|output|
   "view and set some output options"
   |interpreter|
   TREE
   |novar|
    (
\getchunk{outputabbreviate}
\getchunk{outputalgebra}
\getchunk{outputcharacters}
\getchunk{outputfortran}
\getchunk{outputfraction}
\getchunk{outputhtml}
\getchunk{outputlength}
\getchunk{outputmathml}
\getchunk{outputopenmath}
\getchunk{outputscript}
\getchunk{outputscripts}
\getchunk{outputshoweditor}
\getchunk{outputtex}
 ))
```

———-

45.31.1 abbreviate

```
-------------------- The abbreviate Option --------------------

 Description: abbreviate type names

 The abbreviate option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.31.2 defvar $abbreviateTypes

— **initvars** —

```
(defvar |$abbreviateTypes| nil "abbreviate type names")
```

———-

— **outputabbreviate** —

```
    (|abbreviate|
     "abbreviate type names"
     |interpreter|
     LITERALS
     |$abbreviateTypes|
     (|on| |off|)
     |off|)
```

———-

45.31.3 algebra

```
---------------------- The algebra Option ----------------------

 Description: display output in algebraic form

 )set output algebra is used to tell AXIOM to turn algebra-style
  output printing on and off, and where to place the output.  By
  default, the destination for the output is the screen but
  printing is turned off.

Syntax:   )set output algebra <arg>
    where arg can be one of
  on          turn algebra printing on (default state)
  off         turn algebra printing off
  console     send algebra output to screen (default state)
  fp<.fe>     send algebra output to file with file prefix fp
              and file extension .fe. If not given,
              .fe defaults to .spout.

If you wish to send the output to a file, you may need to issue
this command twice: once with on and once with the file name.
For example, to send algebra output to the file polymer.spout,
issue the two commands
```

```
  )set output algebra on
  )set output algebra polymer

The output is placed in the directory from which you invoked
AXIOM or the one you set with the )cd system command.
The current setting is:  On:CONSOLE
```

45.31.4 defvar $algebraFormat

— **initvars** —

```
(defvar |$algebraFormat| t "display output in algebraic form")
```

———-

45.31.5 defvar $algebraOutputFile

— **initvars** —

```
(defvar |$algebraOutputFile| "CONSOLE"
  "where algebra printing goes (enter {\em console} or a pathname)?")
```

———-

— **outputalgebra** —

```
    (|algebra|
     "display output in algebraic form"
     |interpreter|
     FUNCTION
     |setOutputAlgebra|
     (("display output in algebraic form"
       LITERALS
       |$algebraFormat|
       (|off| |on|)
       |on|)
      (break $algebraFormat)
      ("where algebra printing goes (enter {\em console} or a pathname)?"
       FILENAME
       |$algebraOutputFile|
       |chkOutputFileName|
       "console"))
     NIL)
```

———-

45.31.6 defvar $algebraOutputStream

— **initvars** —

```
(defvar |$algebraOutputStream| *standard-output*)
```

———-

45.31.7 defun setOutputAlgebra

[defiostream p972]
[concat p1041]
[describeSetOutputAlgebra p756]
[qcdr p??]
[qcar p??]
[member p1042]
[upcase p??]
[sayKeyedMsg p327]
[shut p972]
[pathnameType p1034]
[pathnameDirectory p1035]
[pathnameName p1034]
[$filep p??]
[make-outstream p971]
[object2String p??]
[$algebraOutputStream p754]
[$algebraOutputFile p753]
[$filep p??]
[$algebraFormat p753]

— **defun setOutputAlgebra** —

```
(defun |setOutputAlgebra| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$algebraOutputStream| |$algebraOutputFile| $filep
  |$algebraFormat|))
   (cond
    ((eq arg '|%initialize%|)
     (setq |$algebraOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$algebraOutputFile| "CONSOLE")
     (setq |$algebraFormat| t))
```

```
((eq arg '|%display%|)
 (if |$algebraFormat|
  (setq label "On:")
  (setq label "Off:"))
 (concat label |$algebraOutputFile|))
((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
 (|describeSetOutputAlgebra|))
(t
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t)
        (|member| fn '(y n ye yes no o on of off console
                |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
   '|ok|)
  (t (setq arg (list fn '|spout|))))
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t))
   (cond
    ((|member| (upcase fn) '(y n ye o of))
     (|sayKeyedMsg| 's2iv0002 '(|algebra| |algebra|)))
    ((|member| (upcase fn) '(no off)) (setq |$algebraFormat| nil))
    ((|member| (upcase fn) '(yes on)) (setq |$algebraFormat| t))
    ((eq (upcase fn) 'console)
     (shut |$algebraOutputStream|)
     (setq |$algebraOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$algebraOutputFile| "CONSOLE"))))
  ((or
    (and (consp arg)
         (progn
          (setq fn (qcar arg))
          (setq tmp1 (qcdr arg))
          (and (consp tmp1)
               (eq (qcdr tmp1) nil)
               (progn (setq ft (qcar tmp1)) t))))
    (and (consp arg)
         (progn (setq fn (qcar arg))
                (setq tmp1 (qcdr arg))
                (and (consp tmp1)
                     (progn (setq ft (qcar tmp1))
                            (setq tmp2 (qcdr tmp1))
                            (and (consp tmp2)
                                 (eq (qcdr tmp2) nil)
                                 (progn
                                  (setq fm (qcar tmp2))
                                  t)))))))
   (when (setq ptype (|pathnameType| fn))
```

```
        (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
        (setq ft ptype))
       (unless fm (setq fm 'a))
       (setq filename ($filep fn ft fm))
       (cond
        ((null filename)
         (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
        ((setq teststream (make-outstream filename 255 0))
         (shut |$algebraOutputStream|)
         (setq |$algebraOutputStream| teststream)
         (setq |$algebraOutputFile| (|object2String| filename))
         (|sayKeyedMsg| 's2iv0004 (list "Algebra" |$algebraOutputFile|)))
        (t (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))))
      (t
       (|sayKeyedMsg| 's2iv0005 nil)
       (|describeSetOutputAlgebra|)))))))
```

———-

45.31.8 defun describeSetOutputAlgebra

[sayBrightly p??]
[setOutputAlgebra p754]

— **defun describeSetOutputAlgebra** —

```
(defun |describeSetOutputAlgebra| ()
 (|sayBrightly| (list
 '|%b| ")set output algebra"
 '|%d| "is used to tell AXIOM to turn algebra-style output"
 '|%l| "printing on and off, and where to place the output.  By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output algebra <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn algebra printing on (default state)"
 '|%l| "  off         turn algebra printing off"
 '|%l| "  console     send algebra output to screen (default state)"
 '|%l| "  fp<.fe>     send algebra output to file with file prefix fp"
 '|%l|
  "              and file extension .fe. If not given, .fe defaults to .spout."
 '|%l|
 '|%l|
 "If you wish to send the output to a file, you may need to issue this command"
 '|%l| "twice: once with"
 '|%b| "on"
 '|%d| "and once with the file name. For example, to send"
```

```
'|%l| "algebra output to the file"
'|%b| "polymer.spout,"
'|%d| "issue the two commands"
'|%l|
'|%l| "  )set output algebra on"
'|%l| "  )set output algebra polymer"
'|%l|
'|%l| "The output is placed in the directory from which you invoked AXIOM or"
'|%l| "the one you set with the )cd system command."
'|%l| "The current setting is: "
'|%b| (|setOutputAlgebra| '|%display%|)
'|%d|)))
```

———-

45.31.9 characters

```
-------------------- The characters Option --------------------

 Description: choose special output character set


 The characters option may be followed by any one of the
 following:

    default
 -> plain

 The current setting is indicated.  This
 option  determines the special characters used for algebraic
 output. This is what the current choice of special characters
 looks like:
   ulc is shown as +          urc is shown as +
   llc is shown as +          lrc is shown as +
   vbar is shown as |         hbar is shown as -
   quad is shown as ?         lbrk is shown as [
   rbrk is shown as ]         lbrc is shown as {
   rbrc is shown as }         ttee is shown as +
   btee is shown as +         rtee is shown as +
   ltee is shown as +         ctee is shown as +
   bslash is shown as \
```

— **outputcharacters** —

```
    (|characters|
     "choose special output character set"
```

```
|interpreter|
FUNCTION
|setOutputCharacters|
NIL
|htSetOutputCharacters|)
```

———-

45.31.10 defun setOutputCharacters

[sayMessage p??]
[bright p??]
[sayBrightly p??]
[concat p1041]
[pname p1039]
[specialChar p970]
[sayAsManyPerLineAsPossible p??]
[qcdr p??]
[qcar p??]
[downcase p??]
[setOutputCharacters p758]
[$specialCharacters p969]
[$plainRTspecialCharacters p968]
[$RTspecialCharacters p968]
[$specialCharacterAlist p969]

— defun setOutputCharacters —

```
(defun |setOutputCharacters| (arg)
 (let (current char s l fn)
 (declare (special |$specialCharacters| |$plainRTspecialCharacters|
   |$RTspecialCharacters| |$specialCharacterAlist|))
    (if (eq arg '|%initialize%|)
     (setq |$specialCharacters| |$plainRTspecialCharacters|)
     (progn
      (setq current
       (cond
        ((eq |$specialCharacters| |$RTspecialCharacters|) "default")
        ((eq |$specialCharacters| |$plainRTspecialCharacters|) "plain")
        (t "unknown")))
      (cond
       ((eq arg '|%display%|) current)
       ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
        (|sayMessage|
         `(" The" ,@(|bright| "characters")
           "option may be followed by any one of the following:"))
```

```
        (dolist (name '("default" "plain"))
         (if (string= (string current) name)
          (|sayBrightly| `(" ->" ,@(|bright| name)))
          (|sayBrightly|  (list "    " name))))
        (terpri)
        (|sayBrightly|
 " The current setting is indicated within the list.  This option determines ")
        (|sayBrightly|
        " the special characters used for algebraic output.  This is what the")
        (|sayBrightly|
         " current choice of special characters looks like:")
        (do ((t1 |$specialCharacterAlist| (CDR t1)) (t2 nil))
           ((or (atom t1)
                (progn (setq t2 (car t1)) nil)
                (progn (progn (setq char (car t2)) t2) nil)) nil)
         (setq s
          (concat "   " (pname char) " is shown as "
           (pname (|specialChar| char))))
          (setq l (cons s l)))
        (|sayAsManyPerLineAsPossible| (reverse l)))
       ((and (consp arg)
             (eq (qcdr arg) NIL)
             (progn (setq fn (qcar arg)) t)
             (setq fn (downcase fn)))
        (cond
         ((eq fn '|default|)
          (setq |$specialCharacters| |$RTspecialCharacters|))
         ((eq fn '|plain|)
          (setq |$specialCharacters| |$plainRTspecialCharacters|))
          (t (|setOutputCharacters| nil))))
       (t (|setOutputCharacters| nil)))))))
```

———-

45.31.11 fortran

```
---------------------- The fortran Option ----------------------

 Description: create output in FORTRAN format

 )set output fortran is used to tell AXIOM to turn FORTRAN-style
  output printing on and off, and where to place the output.  By
  default, the destination for the output is the screen but
  printing is turned off.

Also See: )set fortran

Syntax:   )set output fortran <arg>
```

```
    where arg can be one of
  on          turn FORTRAN printing on
  off         turn FORTRAN printing off (default state)
  console     send FORTRAN output to screen (default state)
  fp<.fe>     send FORTRAN output to file with file prefix
              fp and file extension .fe. If not given,
              .fe defaults to .sfort.

If you wish to send the output to a file, you must issue
this command twice: once with on and once with the file name.
For example, to send FORTRAN output to the file polymer.sfort,
 issue the two commands

  )set output fortran on
  )set output fortran polymer

The output is placed in the directory from which you invoked
AXIOM or the one you set with the )cd system command.
The current setting is:  Off:CONSOLE
```

45.31.12 defvar $fortranFormat

— **initvars** —

```
(defvar |$fortranFormat| nil "create output in FORTRAN format")
```

———-

45.31.13 defvar $fortranOutputFile

— **initvars** —

```
(defvar |$fortranOutputFile| "CONSOLE"
   "where FORTRAN output goes (enter {\em console} or a a pathname)")
```

———-

— **outputfortran** —

```
     (|fortran|
      "create output in FORTRAN format"
      |interpreter|
      FUNCTION
```

```
|setOutputFortran|
(("create output in FORTRAN format"
  LITERALS
  |$fortranFormat|
  (|off| |on|)
  |off|)
 (|break| |$fortranFormat|)
 ("where FORTRAN output goes (enter {\em console} or a a pathname)"
  FILENAME
  |$fortranOutputFile|
  |chkOutputFileName|
  "console"))
NIL)
```

———-

45.31.14 defun setOutputFortran

[defiostream p972]
[concat p1041]
[describeSetOutputFortran p763]
[upcase p**??**]
[qcdr p**??**]
[qcar p**??**]
[member p1042]
[sayKeyedMsg p327]
[shut p972]
[pathnameType p1034]
[pathnameDirectory p1035]
[pathnameName p1034]
[$filep p**??**]
[makeStream p973]
[object2String p**??**]
[$fortranOutputStream p**??**]
[$fortranOutputFile p760]
[$filep p**??**]
[$fortranFormat p760]

— **defun setOutputFortran** —

```
(defun |setOutputFortran| (arg)
 (let (label APPEND quiet tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$fortranOutputStream| |$fortranOutputFile| $filep
    |$fortranFormat|))
  (cond
   ((eq arg '|%initialize%|)
```

```
 (setq |$fortranOutputStream|
  (defiostream '((mode . output) (device . console)) 255 0))
 (setq |$fortranOutputFile| "CONSOLE")
 (setq |$fortranFormat| nil))
((eq arg '|%display%|)
 (if |$fortranFormat|
  (setq label "On:")
  (setq label "Off:"))
 (concat label |$fortranOutputFile|))
((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
 (|describeSetOutputFortran|))
(t
 (DO ()
     ((null (and (listp arg)
                 (|member| (upcase (car arg)) '(append quiet))))
    nil)
  (cond
   ((eq (upcase (car arg)) 'append) (setq append t))
   ((eq (upcase (car arg)) 'quiet) (setq quiet t))
   (t nil))
  (setq arg (cdr arg)))
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t)
        (|member| fn '(Y N YE YES NO O ON OF OFF CONSOLE
          |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
    '|ok|)
  (t (setq arg (list fn '|sfort|))))
 (cond
  ((and (consp arg) (eq (qcdr arg) nil) (progn (setq fn (qcar arg)) t))
   (cond
    ((|member| (upcase fn) '(y n ye o of))
     (|sayKeyedMsg| 's2iv0002 '(fortran |fortran|)))
    ((|member| (upcase fn) '(no off)) (setq |$fortranFormat| nil))
    ((|member| (upcase fn) '(yes on)) (setq |$fortranFormat| t))
    ((eq (upcase fn) 'console)
      (shut |$fortranOutputStream|)
      (setq |$fortranOutputStream|
       (defiostream '((mode . output) (device . console)) 255 0))
      (setq |$fortranOutputFile| "CONSOLE"))))
  ((or
    (and (consp arg)
         (progn
          (setq fn (qcar arg))
          (setq tmp1 (qcdr arg))
          (and (consp tmp1)
               (eq (qcdr tmp1) nil)
               (progn (setq ft (qcar tmp1)) t))))
    (and (consp arg)
```

```
           (progn
            (setq fn (qcar arg))
            (setq tmp1 (qcdr arg))
            (and (consp tmp1)
                 (progn
                  (setq ft (qcar tmp1))
                  (setq tmp2 (qcdr tmp1))
                  (and (consp tmp2)
                       (eq (qcdr tmp2) nil)
                       (progn (setq fm (qcar tmp2)) t)))))))
     (when (setq ptype (|pathnameType| fn))
       (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
       (setq ft ptype))
     (unless fm (setq fm 'a))
     (setq filename ($filep fn ft fm))
     (cond
      ((null filename)
       (|sayKeyedMsg| 'S2IV0003 (list fn ft fm)))
      ((setq teststream (|makeStream| append filename 255 0))
       (SHUT |$fortranOutputStream|)
       (setq |$fortranOutputStream| teststream)
       (setq |$fortranOutputFile| (|object2String| filename))
       (unless quiet
         (|sayKeyedMsg| 'S2IV0004 (list 'fortran |$fortranOutputFile|))))
      ((null quiet)
       (|sayKeyedMsg| 'S2IV0003 (list fn ft fm)))
      (t nil)))
    (t
     (unless quiet (|sayKeyedMsg| 'S2IV0005 nil))
     (|describeSetOutputFortran|)))))))
```

———-

45.31.15 defun describeSetOutputFortran

[sayBrightly p**??**]
[setOutputFortran p761]

— **defun describeSetOutputFortran** —

```
(defun |describeSetOutputFortran| ()
 (|sayBrightly| (list
  '|%b| ")set output fortran"
  '|%d| "is used to tell AXIOM to turn FORTRAN-style output"
  '|%l| "printing on and off, and where to place the output.  By default, the"
  '|%l| "destination for the output is the screen but printing is turned off."
  '|%l|
```

```
  '|%l| "Also See: )set fortran"
  '|%l|
  '|%l| "Syntax:   )set output fortran <arg>"
  '|%l| "    where arg can be one of"
  '|%l| "  on          turn FORTRAN printing on"
  '|%l| "  off         turn FORTRAN printing off (default state)"
  '|%l| "  console     send FORTRAN output to screen (default state)"
  '|%l|
  "  fp<.fe>     send FORTRAN output to file with file prefix fp and file"
  '|%l| "              extension .fe. If not given, .fe defaults to .sfort."
  '|%l|
  '|%l| "If you wish to send the output to a file, you must issue this command"
  '|%l| "twice: once with"
  '|%b| "on"
  '|%d| "and once with the file name. For example, to send"
  '|%l| "FORTRAN output to the file"
  '|%b| "polymer.sfort,"
  '|%d| "issue the two commands"
  '|%l|
  '|%l| "  )set output fortran on"
  '|%l| "  )set output fortran polymer"
  '|%l|
  '|%l| "The output is placed in the directory from which you invoked AXIOM or"
  '|%l| "the one you set with the )cd system command."
  '|%l| "The current setting is: "
  '|%b| (|setOutputFortran| '|%display%|)
  '|%d|)))
```

———-

45.31.16 fraction

```
---------------------- The fraction Option ----------------------

 Description: how fractions are formatted

 The fraction option may be followed by any one of the following:

 -> vertical
    horizontal

 The current setting is indicated.
```

45.31.17 defvar $fractionDisplayType

— **initvars** —

```
(defvar |$fractionDisplayType| '|vertical| "how fractions are formatted")
```

———-

— **outputfraction** —

```
    (|fraction|
     "how fractions are formatted"
     |interpreter|
     LITERALS
     |$fractionDisplayType|
     (|vertical| |horizontal|)
     |vertical|)
```

———-

45.31.18 length

```
---------------------- The length Option ----------------------

 Description: line length of output displays

 The length option may be followed by an integer in the range
 10 to 245 inclusive. The current setting is 77
```

45.31.19 defvar $margin

— **initvars** —

```
(defvar $margin 3)
```

———-

45.31.20 defvar $linelength

— **initvars** —

```
(defvar $linelength 77 "line length of output displays")
```

———-

— outputlength —

```
(|length|
 "line length of output displays"
 |interpreter|
 INTEGER
 $LINELENGTH
 (10 245)
 77)
```

———-

45.31.21 mathml

```
---------------------- The mathml Option -----------------------

 Description: create output in MathML style

 )set output mathml is used to tell AXIOM to turn MathML-style output
printing on and off, and where to place the output.  By default,
the destination for the output is the screen but printing is
turned off.

Syntax:   )set output mathml <arg>
    where arg can be one of
  on          turn MathML printing on
  off         turn MathML printing off (default state)
  console     send MathML output to screen (default state)
  fp<.fe>     send MathML output to file with file prefix fp
              and file extension .fe. If not given,
              .fe defaults to .smml.

If you wish to send the output to a file, you must issue
this command twice: once with on and once with the file name.
For example, to send MathML output to the file polymer.smml,
issue the two commands

  )set output mathml on
  )set output mathml polymer

The output is placed in the directory from which you invoked
AXIOM or the one you set with the )cd system command.
```

```
The current setting is:  Off:CONSOLE
```

45.31.22 defvar $mathmlFormat

— **initvars** —

```
(defvar |$mathmlFormat| nil "create output in MathML format")
```

————-

45.31.23 defvar $mathmlOutputFile

— **initvars** —

```
(defvar |$mathmlOutputFile| "CONSOLE"
   "where MathML output goes (enter {\em console} or a pathname)")
```

————-

— **outputmathml** —

```
     (|mathml|
      "create output in MathML style"
      |interpreter|
      FUNCTION
      |setOutputMathml|
      (("create output in MathML format"
        LITERALS
        |$mathmlFormat|
        (|off| |on|)
        |off|)
       (|break| |$mathmlFormat|)
       ("where MathML output goes (enter {\em console} or a pathname)"
        FILENAME
        |$mathmlOutputFile|
        |chkOutputFileName|
        "console"))
      NIL)
```

————-

45.31.24 defun setOutputMathml

[defiostream p972]
[concat p1041]
[describeSetOutputMathml p770]
[qcdr p??]
[qcar p??]
[member p1042]
[upcase p??]
[sayKeyedMsg p327]
[shut p972]
[pathnameType p1034]
[pathnameDirectory p1035]
[pathnameName p1034]
[$filep p??]
[make-outstream p971]
[object2String p??]
[$mathmlOutputStream p??]
[$mathmlOutputFile p767]
[$mathmlFormat p767]
[$filep p??]

— **defun setOutputMathml** —

```
(defun |setOutputMathml| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$mathmlOutputStream| |$mathmlOutputFile| |$mathmlFormat|
           $filep))
   (cond
    ((eq arg '|%initialize%|)
     (setq |$mathmlOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$mathmlOutputFile| "CONSOLE")
     (setq |$mathmlFormat| nil))
    ((eq arg '|%display%|)
     (if |$mathmlFormat|
      (setq label "On:")
      (setq label "Off:"))
     (concat label |$mathmlOutputFile|))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetOutputMathml|))
    (t
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t)
            (|member| fn '(y n ye yes no o on of off console
                   |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
```

```
    '|ok|)
   (t (setq arg (list fn '|smml|))))
  (cond
   ((and (consp arg)
         (eq (qcdr arg) nil)
         (progn (setq fn (qcar arg)) t))
    (cond
     ((|member| (upcase fn) '(y n ye o of))
      (|sayKeyedMsg| 's2iv0002 '(|MathML| |mathml|)))
     ((|member| (upcase fn) '(no off)) (setq |$mathmlFormat| nil))
     ((|member| (upcase fn) '(yes on)) (setq |$mathmlFormat| t))
     ((eq (upcase fn) 'console)
      (shut |$mathmlOutputStream|)
      (setq |$mathmlOutputStream|
       (defiostream '((mode . output) (device . console)) 255 0))
      (setq |$mathmlOutputFile| "CONSOLE"))))
   ((or
     (and (consp arg)
          (progn
           (setq fn (qcar arg))
           (setq tmp1 (qcdr arg))
           (and (consp tmp1)
                (eq (qcdr tmp1) nil)
                (progn (setq ft (qcar tmp1)) t))))
     (and (consp arg)
          (progn (setq fn (qcar arg))
                 (setq tmp1 (qcdr arg))
                 (and (consp tmp1)
                      (progn
                       (setq ft (qcar tmp1))
                       (setq tmp2 (qcdr tmp1))
                       (and (consp tmp2)
                            (eq (qcdr tmp2) nil)
                            (progn
                             (setq fm (qcar tmp2))
                             t)))))))
     (when (setq ptype (|pathnameType| fn))
      (setq fn
       (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
      (setq ft ptype))
     (unless fm (setq fm 'a))
     (setq filename ($filep fn ft fm))
     (cond
      ((null filename) (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
      ((setq teststream (make-outstream filename 255 0))
       (shut |$mathmlOutputStream|)
       (setq |$mathmlOutputStream| teststream)
       (setq |$mathmlOutputFile| (|object2String| filename))
       (|sayKeyedMsg| 's2iv0004 (list "MathML" |$mathmlOutputFile|)))
      (t (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))))
```

```
      (t
       (|sayKeyedMsg| 's2iv0005 nil)
       (|describeSetOutputMathml|)))))))
```

———-

45.31.25 defun describeSetOutputMathml

[sayBrightly p??]
[setOutputMathml p768]

— defun describeSetOutputMathml —

```
(defun |describeSetOutputMathml| ()
 (|sayBrightly| (LIST
 '|%b| ")set output mathml"
 '|%d| "is used to tell AXIOM to turn MathML-style output"
 '|%l| "printing on and off, and where to place the output.  By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output mathml <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn MathML printing on"
 '|%l| "  off         turn MathML printing off (default state)"
 '|%l| "  console     send MathML output to screen (default state)"
 '|%l| "  fp<.fe>     send MathML output to file with file prefix fp and file"
 '|%l| "              extension .fe. If not given, .fe defaults to .stex."
 '|%l|
 '|%l| "If you wish to send the output to a file, you must issue this command"
 '|%l| "twice: once with"
 '|%b| "on"
 '|%d| "and once with the file name. For example, to send"
 '|%l| "MathML output to the file"
 '|%b| "polymer.smml,"
 '|%d| "issue the two commands"
 '|%l|
 '|%l| "  )set output mathml on"
 '|%l| "  )set output mathml polymer"
 '|%l|
 '|%l| "The output is placed in the directory from which you invoked AXIOM or"
 '|%l| "the one you set with the )cd system command."
 '|%l| "The current setting is: "
 '|%b| (|setOutputMathml| '|%display%|)
 '|%d|)))
```

———-

45.31.26 html

```
---------------------- The html Option -----------------------

 Description: create output in html style

 )set output html is used to tell AXIOM to turn html-style output
printing on and off, and where to place the output.  By default,
the destination for the output is the screen but printing is
turned off.

Syntax:   )set output html <arg>
    where arg can be one of
  on          turn html printing on
  off         turn html printing off (default state)
  console     send html output to screen (default state)
  fp<.fe>     send html output to file with file prefix fp
              and file extension .fe. If not given,
              .fe defaults to .html.

If you wish to send the output to a file, you must issue
this command twice: once with on and once with the file name.
For example, to send MathML output to the file polymer.html,
issue the two commands

  )set output html on
  )set output html polymer

The output is placed in the directory from which you invoked
Axiom or the one you set with the )cd system command.
The current setting is:  Off:CONSOLE
```

45.31.27 defvar $htmlFormat

— **initvars** —

```
(defvar |$htmlFormat| nil "create output in HTML format")
```

———-

45.31.28 defvar $htmlOutputFile

— **initvars** —

```
(defvar |$htmlOutputFile| "CONSOLE"
```

```
"where HTML output goes (enter {\em console} or a pathname)")
```

———-

— **outputhtml** —

```
(|html|
 "create output in HTML style"
 |interpreter|
 FUNCTION
 |setOutputHtml|
 (("create output in HTML format"
   LITERALS
   |$htmlFormat|
   (|off| |on|)
   |off|)
  (|break| |$htmlFormat|)
  ("where HTML output goes (enter {\em console} or a pathname)"
   FILENAME
   |$htmlOutputFile|
   |chkOutputFileName|
   "console"))
 NIL)
```

———-

45.31.29 defun setOutputHtml

[defiostream p972]
[concat p1041]
[describeSetOutputHtml p774]
[qcdr p??]
[qcar p??]
[member p1042]
[upcase p??]
[sayKeyedMsg p327]
[shut p972]
[pathnameType p1034]
[pathnameDirectory p1035]
[pathnameName p1034]
[$filep p??]
[make-outstream p971]
[object2String p??]
[$htmlOutputStream p??]
[$htmlOutputFile p771]

[$htmlFormat p771]
[$filep p??]

— **defun setOutputHtml** —

```
(defun |setOutputHtml| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$htmlOutputStream| |$htmlOutputFile| |$htmlFormat|
          $filep))
   (cond
    ((eq arg '|%initialize%|)
     (setq |$htmlOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$htmlOutputFile| "CONSOLE")
     (setq |$htmlFormat| nil))
    ((eq arg '|%display%|)
     (if |$htmlFormat|
      (setq label "On:")
      (setq label "Off:"))
     (concat label |$htmlOutputFile|))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetOutputHtml|))
    (t
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t)
            (|member| fn '(y n ye yes no o on of off console
                   |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
        '|ok|)
      (t (setq arg (list fn '|smml|))))
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t))
       (cond
        ((|member| (upcase fn) '(y n ye o of))
         (|sayKeyedMsg| 's2iv0002 '(|HTML| |html|)))
        ((|member| (upcase fn) '(no off)) (setq |$htmlFormat| nil))
        ((|member| (upcase fn) '(yes on)) (setq |$htmlFormat| t))
        ((eq (upcase fn) 'console)
         (shut |$htmlOutputStream|)
         (setq |$htmlOutputStream|
          (defiostream '((mode . output) (device . console)) 255 0))
         (setq |$htmlOutputFile| "CONSOLE"))))
      ((or
        (and (consp arg)
             (progn
              (setq fn (qcar arg))
              (setq tmp1 (qcdr arg))
```

```
            (and (consp tmp1)
                 (eq (qcdr tmp1) nil)
                 (progn (setq ft (qcar tmp1)) t))))
       (and (consp arg)
            (progn (setq fn (qcar arg))
                   (setq tmp1 (qcdr arg))
                   (and (consp tmp1)
                        (progn
                         (setq ft (qcar tmp1))
                         (setq tmp2 (qcdr tmp1))
                         (and (consp tmp2)
                              (eq (qcdr tmp2) nil)
                              (progn
                               (setq fm (qcar tmp2))
                               t)))))))
         (when (setq ptype (|pathnameType| fn))
          (setq fn
           (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
          (setq ft ptype))
         (unless fm (setq fm 'a))
         (setq filename ($filep fn ft fm))
         (cond
          ((null filename) (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
          ((setq teststream (make-outstream filename 255 0))
           (shut |$htmlOutputStream|)
           (setq |$htmlOutputStream| teststream)
           (setq |$htmlOutputFile| (|object2String| filename))
           (|sayKeyedMsg| 's2iv0004 (list "HTML" |$htmlOutputFile|)))
          (t (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))))
      (t
       (|sayKeyedMsg| 's2iv0005 nil)
       (|describeSetOutputHtml|)))))
```

———-

45.31.30 defun describeSetOutputHtml

[sayBrightly p**??**]
[setOutputHtml p772]

— **defun describeSetOutputHtml** —

```
(defun |describeSetOutputHtml| ()
 (|sayBrightly| (LIST
 '|%b| ")set output html"
 '|%d| "is used to tell AXIOM to turn HTML-style output"
 '|%l| "printing on and off, and where to place the output.  By default, the"
```

```
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output html <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn HTML printing on"
 '|%l| "  off         turn HTML printing off (default state)"
 '|%l| "  console     send HTML output to screen (default state)"
 '|%l| "  fp<.fe>     send HTML output to file with file prefix fp and file"
 '|%l| "              extension .fe. If not given, .fe defaults to .stex."
 '|%l|
 '|%l| "If you wish to send the output to a file, you must issue this command"
 '|%l| "twice: once with"
 '|%b| "on"
 '|%d| "and once with the file name. For example, to send"
 '|%l| "HTML output to the file"
 '|%b| "polymer.smml,"
 '|%d| "issue the two commands"
 '|%l|
 '|%l| "  )set output html on"
 '|%l| "  )set output html polymer"
 '|%l|
 '|%l| "The output is placed in the directory from which you invoked AXIOM or"
 '|%l| "the one you set with the )cd system command."
 '|%l| "The current setting is: "
 '|%b| (|setOutputHtml| '|%display%|)
 '|%d|)))
```

———-

45.31.31 openmath

```
----------------------- The openmath Option -----------------------

 Description: create output in OpenMath style

 )set output tex is used to tell AXIOM to turn OpenMath output
printing on and off, and where to place the output.  By default,
the destination for the output is the screen but printing is
turned off.

Syntax:   )set output tex <arg>
    where arg can be one of
  on          turn OpenMath printing on
  off         turn OpenMath printing off (default state)
  console     send OpenMath output to screen (default state)
  fp<.fe>     send OpenMath output to file with file prefix fp
              and file extension .fe. If not given,
              .fe defaults to .sopen.
```

```
If you wish to send the output to a file, you must issue
this command twice: once with on and once with the file name.
For example, to send OpenMath output to the file polymer.sopen,
issue the two commands

  )set output openmath on
  )set output openmath polymer

The output is placed in the directory from which you invoked
AXIOM or the one you set with the )cd system command.
The current setting is:  Off:CONSOLE
```

45.31.32 defvar $openMathFormat

— **initvars** —

```
(defvar |$openMathFormat| nil "create output in OpenMath format")
```

———-

45.31.33 defvar $openMathOutputFile

— **initvars** —

```
(defvar |$openMathOutputFile| "CONSOLE"
  "where TeX output goes (enter {\em console} or a pathname)")
```

———-

— **outputopenmath** —

```
    (|openmath|
     "create output in OpenMath style"
     |interpreter|
     FUNCTION
     |setOutputOpenMath|
     (("create output in OpenMath format"
       LITERALS
       |$openMathFormat|
       (|off| |on|)
       |off|)
      (|break| |$openMathFormat|)
```

```
       ("where TeX output goes (enter {\em console} or a pathname)"
        FILENAME
        |$openMathOutputFile|
        |chkOutputFileName|
        "console"))
      NIL)
```

———-

45.31.34 defun setOutputOpenMath

[defiostream p972]
[concat p1041]
[describeSetOutputOpenMath p779]
[qcdr p??]
[qcar p??]
[member p1042]
[upcase p??]
[sayKeyedMsg p327]
[shut p972]
[pathnameType p1034]
[pathnameDirectory p1035]
[pathnameName p1034]
[$filep p??]
[make-outstream p971]
[object2String p??]
[$openMathOutputStream p??]
[$openMathFormat p776]
[$filep p??]
[$openMathOutputFile p776]

— **defun setOutputOpenMath** —

```
(defun |setOutputOpenMath| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$openMathOutputStream| |$openMathFormat| $filep
     |$openMathOutputFile|))
   (cond
    ((eq arg '|%initialize%|)
     (setq |$openMathOutputStream|
      (defiostream  '((mode . output) (device . console)) 255 0))
     (setq |$openMathOutputFile| "CONSOLE")
     (setq |$openMathFormat| NIL))
    ((eq arg '|%display%|)
     (if |$openMathFormat|
      (setq label "On:")
```

```
  (setq label "Off:"))
 (concat label |$openMathOutputFile|))
((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
 (|describeSetOutputOpenMath|))
(t
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t)
        (|member| fn '(y n ye yes no o on of off console
               |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
    '|ok|)
  (t (setq arg (list fn '|som|))))
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t))
   (cond
    ((|member| (upcase fn) '(y n ye o of))
     (|sayKeyedMsg| 's2iv0002 '(|OpenMath| |openmath|)))
    ((|member| (upcase fn) '(no off)) (setq |$openMathFormat| nil))
    ((|member| (upcase fn) '(yes on)) (setq |$openMathFormat| t))
    ((eq (upcase fn) 'console)
     (shut |$openMathOutputStream|)
     (setq |$openMathOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$openMathOutputFile| "CONSOLE"))))
  ((or
    (and (consp arg)
         (progn (setq fn (qcar arg))
                (setq tmp1 (qcdr arg))
                (and (consp tmp1)
                     (eq (qcdr tmp1) nil)
                     (progn (setq ft (qcar tmp1)) t))))
    (and (consp arg)
         (progn
          (setq fn (qcar arg))
          (setq tmp1 (qcdr arg))
          (and (consp tmp1)
               (progn (setq ft (qcar tmp1))
                      (setq tmp2 (qcdr tmp1))
                      (and (consp tmp2)
                           (eq (qcdr tmp2) nil)
                           (progn (setq fm (qcar tmp2)) t)))))))
    (when (setq ptype (|pathnameType| fn))
     (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
     (setq ft ptype))
    (unless fm (setq fm 'a))
    (setq filename ($filep fn ft fm))
    (cond
```

```
        ((null filename)
         (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
        ((setq teststream (make-outstream filename 255 0))
         (shut |$openMathOutputStream|)
         (setq |$openMathOutputStream| teststream)
         (setq |$openMathOutputFile| (|object2String| filename))
         (|sayKeyedMsg| 's2iv0004 (list "OpenMath" |$openMathOutputFile|)))
        (t
         (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))))
      (t
       (|sayKeyedMsg| 's2iv0005 nil)
       (|describeSetOutputOpenMath|)))))))
```

———-

45.31.35 defun describeSetOutputOpenMath

[sayBrightly p**??**]
[setOutputOpenMath p777]

— **defun describeSetOutputOpenMath** —

```
(defun |describeSetOutputOpenMath| ()
 (|sayBrightly| (list
 '|%b| ")set output openmath"
 '|%d| "is used to tell AXIOM to turn OpenMath output"
 '|%l| "printing on and off, and where to place the output.  By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output openmath <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn OpenMath printing on"
 '|%l| "  off         turn OpenMath printing off (default state)"
 '|%l| "  console     send OpenMath output to screen (default state)"
 '|%l|
 "  fp<.fe>     send OpenMath output to file with file prefix fp and file"
 '|%l| "              extension .fe. If not given, .fe defaults to .som."
 '|%l|
 '|%l| "If you wish to send the output to a file, you must issue this command"
 '|%l| "twice: once with"
 '|%b| "on"
 '|%d| "and once with the file name. For example, to send"
 '|%l| "OpenMath output to the file"
 '|%b| "polymer.som,"
 '|%d| "issue the two commands"
 '|%l|
 '|%l| "  )set output openmath on"
```

```
 '|%l| "  )set output openmath polymer"
 '|%l|
 '|%l| "The output is placed in the directory from which you invoked AXIOM or"
 '|%l| "the one you set with the )cd system command."
 '|%l| "The current setting is: "
 '|%b| (|setOutputOpenMath| '|%display%|)
 '|%d|)))
```

———-

45.31.36 script

```
---------------------- The script Option ----------------------

 Description: display output in SCRIPT formula format

 )set output script is used to tell AXIOM to turn IBM Script
 formula-style output printing on and off, and where to place
 the output.  By default, the destination for the output is the
 screen but printing is turned off.

Syntax:   )set output script <arg>
    where arg can be one of
  on      turn IBM Script formula printing on
  off     turn IBM Script formula printing off
          (default state)
  console send IBM Script formula output to screen
          (default state)
  fp<.fe> send IBM Script formula output to file with file
          prefix fp and file extension .fe. If not given,
          .fe defaults to .sform.

If you wish to send the output to a file, you must issue
this command twice: once with on and once with the file
name. For example, to send IBM Script formula output to
the file polymer.sform, issue the two commands

  )set output script on
  )set output script polymer

The output is placed in the directory from which you
invoked AXIOM or the one you set with the )cd system command.
The current setting is:  Off:CONSOLE
```

45.31.37 defvar $formulaFormat

— **initvars** —

```
(defvar |$formulaFormat| nil "display output in SCRIPT format")
```

———-

45.31.38 defvar $formulaOutputFile

— **initvars** —

```
(defvar |$formulaOutputFile| "CONSOLE"
    "where script output goes (enter {\em console} or a a pathname)")
```

———-

— **outputscript** —

```
     (|script|
      "display output in SCRIPT formula format"
      |interpreter|
      FUNCTION
      |setOutputFormula|
      (("display output in SCRIPT format"
        LITERALS
        |$formulaFormat|
        (|off| |on|)
        |off|)
       (|break| |$formulaFormat|)
       ("where script output goes (enter {\em console} or a a pathname)"
        FILENAME
        |$formulaOutputFile|
        |chkOutputFileName|
        "console"))
      NIL)
```

———-

45.31.39 defun setOutputFormula

[defiostream p972]
[concat p1041]

[describeSetOutputFormula p784]
[qcdr p??]
[qcar p??]
[member p1042]
[upcase p??]
[sayKeyedMsg p327]
[shut p972]
[pathnameType p1034]
[pathnameDirectory p1035]
[pathnameName p1034]
[$filep p??]
[make-outstream p971]
[object2String p??]
[$formulaOutputStream p??]
[$formulaOutputFile p781]
[$filep p??]
[$formulaFormat p781]

— **defun setOutputFormula** —

```
(defun |setOutputFormula| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
 (declare (special |$formulaOutputStream| |$formulaOutputFile| $filep
    |$formulaFormat|))
   (cond
    ((eq arg '|%initialize%|)
     (setq |$formulaOutputStream|
      (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$formulaOutputFile| "CONSOLE")
     (setq |$formulaFormat| nil))
    ((eq arg '|%display%|)
     (if |$formulaFormat|
      (setq label "On:")
      (setq label "Off:"))
     (concat label |$formulaOutputFile|))
    ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetOutputFormula|))
    (t
     (cond
      ((and (consp arg)
            (eq (qcdr arg) nil)
            (progn (setq fn (qcar arg)) t)
            (|member| fn '(y n ye yes no o on of off console
                      |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
        '|ok|)
      (t (setq arg (list fn '|sform|))))
     (cond
      ((and (consp arg)
```

```
      (eq (qcdr arg) nil)
      (progn (setq fn (qcar arg)) t))
 (cond
  ((|member| (upcase fn) '(y n ye o of))
   (|sayKeyedMsg| 's2iv0002 '(|script| |script|)))
  ((|member| (upcase fn) '(no off)) (setq |$formulaFormat| nil))
  ((|member| (upcase fn) '(yes on)) (setq |$formulaFormat| t))
  ((eq (upcase fn) 'console)
   (SHUT |$formulaOutputStream|)
   (setq |$formulaOutputStream|
    (defiostream '((mode . output) (device . console)) 255 0))
   (setq |$formulaOutputFile| "CONSOLE"))))
((or
  (and (consp arg)
       (progn (setq fn (qcar arg))
              (setq tmp1 (qcdr arg))
              (and (consp tmp1)
                   (eq (qcdr tmp1) nil)
                   (progn (setq ft (qcar tmp1)) t))))
  (and (consp arg)
       (progn (setq fn (qcar arg))
              (setq tmp1 (qcdr arg))
              (and (consp tmp1)
                   (progn (setq ft (qcar tmp1))
                          (setq tmp2 (qcdr tmp1))
                          (and (consp tmp2)
                               (eq (qcdr tmp2) nil)
                               (progn
                                (setq fm (qcar tmp2)) t)))))))
  (if (setq ptype (|pathnameType| fn))
    (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
    (setq ft ptype))
  (unless fm (setq fm 'a))
  (setq filename ($filep fn ft fm))
  (cond
   ((null filename) (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))
   ((setq teststream (make-outstream filename 255 0))
    (shut |$formulaOutputStream|)
    (setq |$formulaOutputStream| teststream)
    (setq |$formulaOutputFile| (|object2String| filename))
    (|sayKeyedMsg| 's2iv0004
     (list "IBM Script formula" |$formulaOutputFile| )))
   (t
    (|sayKeyedMsg| 's2iv0003 (list fn ft fm)))))
(t
 (|sayKeyedMsg| 's2iv0005 nil)
 (|describeSetOutputFormula|))))))))
```

———-

45.31.40 defun describeSetOutputFormula

[sayBrightly p**??**]
[setOutputFormula p781]

— defun describeSetOutputFormula —

```
(defun |describeSetOutputFormula| ()
 (|sayBrightly| (list
 '|%b| ")set output script"
 '|%d| "is used to tell AXIOM to turn IBM Script formula-style"
 '|%l|
 "output printing on and off, and where to place the output.  By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output script <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn IBM Script formula printing on"
 '|%l| "  off         turn IBM Script formula printing off (default state)"
 '|%l| "  console     send IBM Script formula output to screen (default state)"
 '|%l|
 "  fp<.fe>     send IBM Script formula output to file with file prefix fp"
 '|%l|
 "              and file extension .fe. If not given, .fe defaults to .sform."
 '|%l|
 '|%l| "If you wish to send the output to a file, you must issue this command"
 '|%l| "twice: once with"
 '|%b| "on"
 '|%d| "and once with the file name. For example, to send"
 '|%l| "IBM Script formula output to the file"
 '|%b| "polymer.sform,"
 '|%d| "issue the two commands"
 '|%l|
 '|%l| "  )set output script on"
 '|%l| "  )set output script polymer"
 '|%l|
 '|%l| "The output is placed in the directory from which you invoked AXIOM or"
 '|%l| "the one you set with the )cd system command."
 '|%l| "The current setting is: "
 '|%b| (|setOutputFormula| '|%display%|)
 '|%d|)))
```

———-

45.31.41 scripts

```
--------------------- The scripts Option ----------------------
```

```
 Description: show subscripts,... linearly

 The scripts option may be followed by any one of the following:

    yes
    no

 The current setting is indicated.
```

45.31.42 defvar $linearFormatScripts

— **initvars** —

```
(defvar |$linearFormatScripts| nil "show subscripts,... linearly")
```

———-

— **outputscripts** —

```
     (|scripts|
      "show subscripts,... linearly"
      |interpreter|
      LITERALS
      |$linearFormatScripts|
      (|on| |off|)
      |off|)
```

———-

45.31.43 showeditor

```
-------------------- The showeditor Option --------------------

 Description: view output of )show in editor

 The showeditor option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.31.44 defvar $useEditorForShowOutput

— **initvars** —

```
(defvar |$useEditorForShowOutput| nil "view output of )show in editor")
```

————-

— **outputshoweditor** —

```
    (|showeditor|
     "view output of )show in editor"
     |interpreter|
     LITERALS
     |$useEditorForShowOutput|
     (|on| |off|)
     |off|)
```

————-

45.31.45 tex

```
----------------------- The tex Option -------------------------

 Description: create output in TeX style

 )set output tex is used to tell AXIOM to turn TeX-style output
printing on and off, and where to place the output.  By default,
the destination for the output is the screen but printing is
turned off.

Syntax:   )set output tex <arg>
    where arg can be one of
  on          turn TeX printing on
  off         turn TeX printing off (default state)
  console     send TeX output to screen (default state)
  fp<.fe>     send TeX output to file with file prefix fp
              and file extension .fe. If not given,
              .fe defaults to .stex.

If you wish to send the output to a file, you must issue
this command twice: once with on and once with the file name.
For example, to send TeX output to the file polymer.stex,
issue the two commands
```

```
  )set output tex on
  )set output tex polymer

The output is placed in the directory from which you invoked
AXIOM or the one you set with the )cd system command.
The current setting is:  Off:CONSOLE
```

45.31.46 defvar $texFormat

— **initvars** —

```
(defvar |$texFormat| nil "create output in TeX format")
```

———-

45.31.47 defvar $texOutputFile

— **initvars** —

```
(defvar |$texOutputFile| "CONSOLE"
    "where TeX output goes (enter {\em console} or a pathname)")
```

———-

— **outputtex** —

```
     (|tex|
      "create output in TeX style"
      |interpreter|
      FUNCTION
      |setOutputTex|
      (("create output in TeX format"
        LITERALS
        |$texFormat|
        (|off| |on|)
        |off|)
       (|break| |$texFormat|)
       ("where TeX output goes (enter {\em console} or a pathname)"
        FILENAME
        |$texOutputFile|
        |chkOutputFileName|
        "console"))
      NIL)
```

———-

45.31.48 defun setOutputTex

[defiostream p972]
[concat p1041]
[describeSetOutputTex p790]
[qcdr p??]
[qcar p??]
[member p1042]
[upcase p??]
[sayKeyedMsg p327]
[shut p972]
[pathnameType p1034]
[pathnameDirectory p1035]
[pathnameName p1034]
[$filep p??]
[make-outstream p971]
[object2String p??]
[$texOutputStream p??]
[$texOutputFile p787]
[$texFormat p787]
[$filep p??]

— **defun setOutputTex** —

```
(defun |setOutputTex| (arg)
 (let (label tmp1 tmp2 ptype fn ft fm filename teststream)
  (declare (special |$texOutputStream| |$texOutputFile| |$texFormat| $filep))
  (cond
   ((eq arg '|%initialize%|)
    (setq |$texOutputStream|
     (defiostream '((mode . output) (device . console)) 255 0))
    (setq |$texOutputFile| "CONSOLE")
    (setq |$texFormat| nil))
   ((eq arg '|%display%|)
    (if |$texFormat|
     (setq label "On:")
     (setq label "Off:"))
    (concat label |$texOutputFile|))
   ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
    (|describeSetOutputTex|))
   (t
    (cond
     ((and (consp arg)
           (eq (qcdr arg) nil)
           (progn (setq fn (qcar arg)) t)
```

```
        (|member| fn '(y n ye yes no o on of off console
              |y| |n| |ye| |yes| |no| |o| |on| |of| |off| |console|)))
    '|ok|)
  (t (setq arg (list fn '|stex| nil))))
 (cond
  ((and (consp arg)
        (eq (qcdr arg) nil)
        (progn (setq fn (qcar arg)) t))
   (cond
    ((|member| (upcase fn) '(y n ye o of))
     (|sayKeyedMsg| 's2iv0002 '(|TeX| |tex|)))
    ((|member| (upcase fn) '(no off)) (setq |$texFormat| nil))
    ((|member| (upcase fn) '(yes on)) (setq |$texFormat| t))
    ((eq (upcase fn) 'console)
     (shut |$texOutputStream|)
     (setq |$texOutputStream|
       (defiostream '((mode . output) (device . console)) 255 0))
     (setq |$texOutputFile| "CONSOLE"))))
  ((or
    (and (consp arg)
         (progn (setq fn (qcar arg))
                (setq tmp1 (qcdr arg))
                (and (consp tmp1)
                     (eq (qcdr tmp1) nil)
                     (progn (setq ft (qcar tmp1)) t))))
   (and (consp arg)
        (progn (setq fn (qcar arg))
               (setq tmp1 (qcdr arg))
               (and (consp tmp1)
               (progn (setq ft (qcar tmp1))
                      (setq tmp2 (qcdr tmp1))
                      (and (consp tmp2)
                           (eq (qcdr tmp2) nil)
                           (progn (setq fm (qcar tmp2)) t)))))))
   (when (setq ptype (|pathnameType| fn))
    (setq fn (concat (|pathnameDirectory| fn) (|pathnameName| fn)))
    (setq ft ptype))
   (unless fm (setq fm 'A))
   (setq filename ($filep fn ft fm))
   (cond
    ((null filename) (|sayKeyedMsg| 's2iv0003 (list fn ft fm )))
    ((setq teststream (make-outstream filename 255 0))
     (shut |$texOutputStream|)
     (setq |$texOutputStream| teststream)
     (setq |$texOutputFile| (|object2String| filename))
     (|sayKeyedMsg| 's2iv0004 (list "TeX" |$texOutputFile|)))
    (t (|sayKeyedMsg| 'S2IV0003 (list fn ft fm )))))
  (t
   (|sayKeyedMsg| 's2iv0005 nil)
   (|describeSetOutputTex|)))))))
```

———-

45.31.49 defun describeSetOutputTex

[sayBrightly p**??**]
[setOutputTex p788]

— **defun describeSetOutputTex** —

```
(defun |describeSetOutputTex| ()
 (|sayBrightly| (list
 '|%b| ")set output tex"
 '|%d| "is used to tell AXIOM to turn TeX-style output"
 '|%l| "printing on and off, and where to place the output.  By default, the"
 '|%l| "destination for the output is the screen but printing is turned off."
 '|%l|
 '|%l| "Syntax:   )set output tex <arg>"
 '|%l| "    where arg can be one of"
 '|%l| "  on          turn TeX printing on"
 '|%l| "  off         turn TeX printing off (default state)"
 '|%l| "  console     send TeX output to screen (default state)"
 '|%l| "  fp<.fe>     send TeX output to file with file prefix fp and file"
 '|%l| "              extension .fe. If not given, .fe defaults to .stex."
 '|%l|
 '|%l| "If you wish to send the output to a file, you must issue this command"
 '|%l| "twice: once with"
 '|%b| "on"
 '|%d| "and once with the file name. For example, to send"
 '|%l| "TeX output to the file"
 '|%b| "polymer.stex,"
 '|%d| "issue the two commands"
 '|%l|
 '|%l| "  )set output tex on"
 '|%l| "  )set output tex polymer"
 '|%l|
 '|%l| "The output is placed in the directory from which you invoked AXIOM or"
 '|%l| "the one you set with the )cd system command."
 '|%l| "The current setting is: "
 '|%b| (|setOutputTex| '|%display%|)
 '|%d|)))
```

———-

45.32 quit

```
---------------------- The quit Option -----------------------

 Description: protected or unprotected quit

 The quit option may be followed by any one of the following:

    protected
 -> unprotected

 The current setting is indicated.
```

45.32.1 defvar $quitCommandType

— **initvars** —

```
(defvar |$quitCommandType| '|protected| "protected or unprotected quit")
```

———-

— **quit** —

```
  (|quit|
   "protected or unprotected quit"
   |interpreter|
   LITERALS
   |$quitCommandType|
   (|protected| |unprotected|)
   |protected|)
```

———-

45.33 streams

```
              Current Values of  streams  Variables

Variable     Description                           Current Value
-----------------------------------------------------------------
calculate    specify number of elements to calculate    10
showall      display all stream elements computed       off
```

— **streams** —

```
  (|streams|
   "set some options for working with streams"
   |interpreter|
   TREE
   |novar|
   (
\getchunk{streamscalculate}
\getchunk{streamsshowall}
  ))
```

———-

45.33.1 calculate

```
-------------------- The calculate Option ---------------------

 Description: specify number of elements to calculate

   )set streams calculate is used to tell AXIOM how many elements
    of a stream to calculate when a computation uses the stream.
    The value given after calculate must either be the word all
    or a positive integer.

      The current setting is 10 .
```

45.33.2 defvar $streamCount

— **initvars** —

```
(defvar |$streamCount| 10
    "number of initial stream elements you want calculated")
```

———-

— **streamscalculate** —

```
     (|calculate|
      "specify number of elements to calculate"
      |interpreter|
      FUNCTION
      |setStreamsCalculate|
```

```
      (("number of initial stream elements you want calculated"
       INTEGER
       |$streamCount|
       (0 NIL)
       10))
      NIL)
```

———-

45.33.3 defun setStreamsCalculate

[object2String p??]
[describeSetStreamsCalculate p793]
[sayMessage p??]
[bright p??]
[terminateSystemCommand p448]
[$streamCount p792]

— **defun setStreamsCalculate** —

```
(defun |setStreamsCalculate| (arg)
 (let (n)
  (declare (special |$streamCount|))
  (cond
   ((eq arg '|%initialize%|) (setq |$streamCount| 10))
   ((eq arg '|%display%|) (|object2String| |$streamCount|))
   ((or (null arg) (eq arg '|%describe%|) (eq (car arg) '?))
     (|describeSetStreamsCalculate|))
   (t
     (setq n (car arg))
     (cond
      ((and (not (eq n '|all|)) (or (null (integerp n)) (minusp n)))
        (|sayMessage|
         `("Your value of" ,@(|bright| n) "is invalid because ..."))
        (|describeSetStreamsCalculate|)
        (|terminateSystemCommand|))
      (t (setq |$streamCount| n)))))))
```

———-

45.33.4 defun describeSetStreamsCalculate

[sayKeyedMsg p327]
[$streamCount p792]

— defun describeSetStreamsCalculate —

```
(defun |describeSetStreamsCalculate| ()
 (declare (special |$streamCount|))
 (|sayKeyedMsg| 's2iv0001 (list |$streamCount|)))
```

———-

45.33.5 showall

```
---------------------- The showall Option ----------------------

 Description: display all stream elements computed

 The showall option may be followed by any one of the following:

    on
 -> off

 The current setting is indicated.
```

45.33.6 defvar $streamsShowAll

— initvars —

```
(defvar |$streamsShowAll| nil "display all stream elements computed")
```

———-

— streamsshowall —

```
    (|showall|
     "display all stream elements computed"
     |interpreter|
     LITERALS
     |$streamsShowAll|
     (|on| |off|)
     |off|)
```

———-

45.34 system

```
             Current Values of  system  Variables

Variable     Description                              Current Value
-----------------------------------------------------------------
functioncode show gen. LISP for functions when compiled off
optimization show optimized LISP code                 off
prettyprint  prettyprint BOOT func's as they compile  off
```

— **system** —

```
  (|system|
   "set some system development variables"
   |development|
   TREE
   |novar|
   (
\getchunk{systemfunctioncode}
\getchunk{systemoptimization}
\getchunk{systemprettyprint}
   ))
```

————-

45.34.1 functioncode

```
------------------- The functioncode Option -------------------

 Description: show gen. LISP for functions when compiled

 The functioncode option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.34.2 defvar $reportCompilation

— **initvars** —

```
(defvar |$reportCompilation| nil "show gen. LISP for functions when compiled")
```

———-

— **systemfunctioncode** —

```
(|functioncode|
 "show gen. LISP for functions when compiled"
 |development|
 LITERALS
 |$reportCompilation|
 (|on| |off|)
 |off|)
```

———-

45.34.3 optimization

```
------------------- The optimization Option --------------------

 Description: show optimized LISP code

 The optimization option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.34.4 defvar $reportOptimization

— **initvars** —

```
(defvar |$reportOptimization| nil "show optimized LISP code")
```

———-

— **systemoptimization** —

```
(|optimization|
```

```
      "show optimized LISP code"
      |development|
      LITERALS
      |$reportOptimization|
      (|on| |off|)
      |off|)
```

———-

45.34.5 prettyprint

```
------------------- The prettyprint Option --------------------

 Description: prettyprint BOOT func's as they compile

 The prettyprint option may be followed by any one of the
 following:

    on
 -> off

 The current setting is indicated.
```

45.34.6 defvar $prettyprint

— **initvars** —

```
(defvar $prettyprint t "prettyprint BOOT func's as they compile")
```

———-

— **systemprettyprint** —

```
    (|prettyprint|
     "prettyprint BOOT func's as they compile"
     |development|
     LITERALS
     $prettyprint
     (|on| |off|)
     |on|)
```

———-

45.35 userlevel

```
-------------------- The userlevel Option ---------------------

 Description: operation access level of system user

 The userlevel option may be followed by any one of the
 following:

    interpreter
    compiler
 -> development

 The current setting is indicated.
```

45.35.1 defvar $UserLevel

— **initvars** —

```
(defvar |$UserLevel| '|development| "operation access level of system user")
```

———-

— **userlevel** —

```
  (|userlevel|
   "operation access level of system user"
   |interpreter|
   LITERALS
   |$UserLevel|
   (|interpreter| |compiler| |development|)
   |development|)
```

———-

— **initvars** —

```
(defvar |$setOptions| '(
\getchunk{breakmode}
\getchunk{compile}
\getchunk{debug}
\getchunk{expose}
\getchunk{functions}
```

```
\getchunk{fortran}
\getchunk{kernel}
\getchunk{hyperdoc}
\getchunk{help}
\getchunk{history}
\getchunk{messages}
\getchunk{naglink}
\getchunk{output}
\getchunk{quit}
\getchunk{streams}
\getchunk{system}
\getchunk{userlevel}
 ))
```

———-

45.35.2 defvar $setOptionNames

— **initvars** —

```
(defvar |$setOptionNames| (mapcar #'car |$setOptions|))
```

———-

— **postvars** —

```
(eval-when (eval load)
 (|initializeSetVariables| |$setOptions|))
```

———-

45.36 Set code

45.36.1 defun set

[set1 p800]
[$setOptions p??]

— **defun set** —

```
(defun |set| (l)
```

```
(declare (special |$setOptions|))
(|set1| l |$setOptions|))
```

———-

45.36.2 defun set1

This function will be called with the top level arguments to)set. For instance, given the command

```
)set break break
```

this function gets

```
(set1 (|break| |break|) ....)
```

and given the command

```
)set mes auto off
```

this function gets

```
(set1 (|mes| |auto| |off|) ....)
```

which, because "message" is a TREE, generates the recursive call:

```
(set1 (|auto| |off|) <the message subtree>)
```

The "autoload" subtree is a FUNCTION (printLoadMessages), which gets called with `%describe%`
[displaySetVariableSettings p649]
[seq p**??**]
[exit p**??**]
[selectOption p475]
[downcase p**??**]
[lassoc p**??**]
[satisfiesUserLevel p447]
[sayKeyedMsg p327]
[poundsign p**??**]
[displaySetOptionInformation p647]
[kdr p**??**]
[sayMSG p329]
[sayMessage p**??**]
[bright p**??**]
[object2String p**??**]
[translateYesNo2TrueFalse p650]

[use-fast-links p??]
[literals p??]
[tree p??]
[set1 p800]
[$setOptionNames p799]
[$UserLevel p798]
[$displaySetValue p739]

— **defun set1** —

```
(defun |set1| (l settree)
 (let (|$setOptionNames| arg setdata st setfunarg num upperlimit arg2)
  (declare (special |$setOptionNames| |$UserLevel| |$displaySetValue|))
  (cond
   ((null l) (|displaySetVariableSettings| settree '||))
   (t
    (setq |$setOptionNames|
      (do ((t1 settree (cdr t1)) t0 (x nil))
          ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
       (seq
        (exit
         (setq t0 (cons (elt x 0) t0))))))
    (setq arg
     (|selectOption| (downcase (car l)) |$setOptionNames| '|optionError|))
    (setq setdata (cons arg (lassoc arg settree)))
    (cond
     ((null (|satisfiesUserLevel| (third setdata)))
      (|sayKeyedMsg| 's2iz0007 (list |$UserLevel| "set option" nil)))
     ((eql 1 (|#| l)) (|displaySetOptionInformation| arg setdata))
     (t
      (setq st (fourth setdata))
      (case (fourth setdata)
       (function
        (setq setfunarg
         (if (eq (elt l 1) 'default)
          '|%initialize%|
          (kdr l)))
        (if (functionp (fifth setdata))
          (funcall (fifth setdata) setfunarg)
          (|sayMSG| (concatenate 'string "   Function not implemented. "
           (string (fifth setdata)))))
        (when |$displaySetValue|
         (|displaySetOptionInformation| arg setdata))
        NIL)
       (string
        (setq arg2 (elt l 1))
        (cond
         ((eq arg2 'default) (set (fifth setdata) (seventh setdata)))
         (arg2 (set (fifth setdata) arg2))
```

```
  (t nil))
 (when (or |$displaySetValue| (null arg2))
  (|displaySetOptionInformation| arg setdata))
 NIL)
(integer
 (setq arg2
  (progn
   (setq num (elt l 1))
   (cond
    ((and (integerp num)
          (>= num (elt (sixth setdata) 0))
          (or (null (setq upperlimit (elt (sixth setdata) 1)))
              (<= num upperlimit)))
     num)
    (t
     (|selectOption|
      (elt l 1)
      (cons '|default| (sixth setdata)) nil)))))
 (cond
  ((eq arg2 'default) (set (fifth setdata) (seventh setdata)))
  (arg2 (set (fifth setdata) arg2))
  (t nil))
 (cond
  ((or |$displaySetValue| (null arg2))
   (|displaySetOptionInformation| arg setdata)))
 (cond
  ((null arg2)
   (|sayMessage|
    `(" Your value" ,@(|bright| (|object2String| (elt l 1)))
      "is not among the valid choices.")))
  (t nil)))
(literals
 (cond
  ((setq arg2
   (|selectOption| (elt l 1)
    (cons '|default| (sixth setdata)) nil))
   (cond
    ((eq arg2 'default)
     (set (fifth setdata)
       (|translateYesNo2TrueFalse| (seventh setdata))))
    (t
     (cond ((eq arg2 '|nobreak|)
       #+:GCL (si::use-fast-links t)))
     (cond
      ((eq arg2 '|fastlinks|)
       #+:GCL (si::use-fast-links nil)
       (setq arg2 '|break|)))
     (set (fifth setdata) (|translateYesNo2TrueFalse| arg2))))))
 (when (or |$displaySetValue| (null arg2))
   (|displaySetOptionInformation| arg setdata))
```

```
 (cond
  ((null arg2)
   (|sayMessage|
    (cons " Your value"
     (append (|bright| (|object2String| (elt l 1)))
      (cons "is not among the valid choices." nil)))))
  (t nil)))
(tree (|set1| (kdr l) (sixth setdata)) nil)
(t
 (|sayMessage|
  `("Cannot handle set tree node type" ,@(|bright| st) |yet|))
 nil))))))))
```

———-

Chapter 46

)show help page Command

46.1 show help page man page

— show.help —

```
====================================================================
A.22.  )show
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )show nameOrAbbrev
  - )show nameOrAbbrev )operations
  - )show nameOrAbbrev )attributes

Command Description:
This command displays information about AXIOM domain, package and category
constructors. If no options are given, the )operations option is assumed. For
example,

)show POLY
)show POLY )operations
)show Polynomial
)show Polynomial )operations

each display basic information about the Polynomial domain constructor and
then provide a listing of operations. Since Polynomial requires a Ring (for
example, Integer) as argument, the above commands all refer to a unspecified
ring R. In the list of operations, $ means Polynomial(R).
```

```
The basic information displayed includes the signature of the constructor
(the name and arguments), the constructor abbreviation, the exposure status
of the constructor, and the name of the library source file for the
constructor.

If operation information about a specific domain is wanted, the full or
abbreviated domain name may be used. For example,

)show POLY INT
)show POLY INT )operations
)show Polynomial Integer
)show Polynomial Integer )operations

are among the combinations that will display the operations exported by the
domain Polynomial(Integer) (as opposed to the general domain constructor
Polynomial). Attributes may be listed by using the )attributes option.

Also See:
o )display
o )set
o )what
```

———-

[1]

46.1.1 defun The)show command

[showSpad2Cmd p806]

— **defun show** —

```
(defun |show| (arg) (|showSpad2Cmd| arg))
```

———-

46.1.2 defun The internal)show command

[member p1042]
[helpSpad2Cmd p568]
[sayKeyedMsg p327]
[qcar p**??**]
[reportOperations p807]
[$showOptions p**??**]

[1] "display" (30.2.1 p 531) "set" (45.36.1 p 799) "what" (53.1.2 p 929)

[$e p??]
[$env p??]
[$InteractiveFrame p??]
[$options p??]

— **defun showSpad2Cmd** —

```
(defun |showSpad2Cmd| (arg)
 (let (|$showOptions| |$e| |$env| constr)
 (declare (special |$showOptions| |$e| |$env| |$InteractiveFrame| |$options|))
  (if (equal arg (list nil))
   (|helpSpad2Cmd| '(|show|))
   (progn
    (setq |$showOptions| '(|attributes| |operations|))
    (unless |$options| (setq |$options| '((|operations|))))
    (setq |$e| |$InteractiveFrame|)
    (setq |$env| |$InteractiveFrame|)
    (cond
     ((and (consp arg) (eq (qcdr arg) nil) (progn (setq constr (qcar arg)) t))
      (cond
       ((|member| constr '(|Union| |Record| |Mapping|))
        (cond
         ((eq constr '|Record|)
           (|sayKeyedMsg| 'S2IZ0044R
            (list constr ")show Record(a: Integer, b: String)" )))
         ((eq constr '|Mapping|) (|sayKeyedMsg| 'S2IZ0044M nil))
         (t
          (|sayKeyedMsg| 'S2IZ0045T
            (list constr ")show Union(a: Integer, b: String)" )))
          (|sayKeyedMsg| 'S2IZ0045U
            (list constr ")show Union(Integer, String)" ))))
       ((and (consp constr) (eq (qcar constr) '|Mapping|))
              (|sayKeyedMsg| 'S2IZ0044M nil))
       (t (|reportOperations| constr constr))))
     (t (|reportOperations| arg arg)))))))
```

———-

46.1.3 defun reportOperations

[sayBrightly p??]
[bright p??]
[sayKeyedMsg p327]
[qcar p??]
[isNameOfType p??]
[isDomainValuedVariable p949]
[reportOpsFromUnitDirectly0 p813]

[opOf p**??**]
[unabbrev p**??**]
[reportOpsFromLisplib0 p809]
[evaluateType p906]
[mkAtree p**??**]
[removeZeroOneDestructively p**??**]
[isType p**??**]
[$env p**??**]
[$eval p**??**]
[$genValue p53]
[$quadSymbol p**??**]
[$doNotAddEmptyModeIfTrue p**??**]

— defun reportOperations —

```
(defun |reportOperations| (oldArg u)
 (let (|$env| |$eval| |$genValue| |$doNotAddEmptyModeIfTrue|
               tmp1 v unitForm tree unitFormp)
 (declare (special |$env| |$eval| |$genValue| |$quadSymbol|
                   |$doNotAddEmptyModeIfTrue|))
  (setq |$env| (list (list nil)))
  (setq |$eval| t)
  (setq |$genValue| t)
  (when u
   (setq |$doNotAddEmptyModeIfTrue| t)
   (cond
    ((equal u |$quadSymbol|)
     (|sayBrightly|
       (cons "   mode denotes" (append (|bright| "any") (list '|type|)))))
    ((eq u '%)
     (|sayKeyedMsg| 'S2IZ0063 nil)
     (|sayKeyedMsg| 'S2IZ0064 nil))
    ((and (null (and (consp u) (eq (qcar u) '|Record|)))
          (null (and (consp u) (eq (qcar u) '|Union|)))
          (null (|isNameOfType| u))
          (null (and (consp u)
                     (eq (qcar u) '|typeOf|)
                     (progn
                      (setq tmp1 (qcdr u))
                      (and (consp tmp1) (eq (qcdr tmp1) nil))))))
     (when (atom oldArg) (setq oldArg (list oldArg)))
     (|sayKeyedMsg| 'S2IZ0063 nil)
     (dolist (op oldArg)
       (|sayKeyedMsg| 'S2IZ0062 (list (|opOf| op)))))
    ((setq v (|isDomainValuedVariable| u)) (|reportOpsFromUnitDirectly0| v))
    (t
      (if (atom u)
       (setq unitForm (|opOf| (|unabbrev| u)))
       (setq unitForm (|unabbrev| u)))
```

```
(if (atom unitForm)
 (|reportOpsFromLisplib0| unitForm u)
 (progn
  (setq unitFormp (|evaluateType| unitForm))
  (setq tree (|mkAtree| (|removeZeroOneDestructively| unitForm)))
  (if (setq unitFormp (|isType| tree))
    (|reportOpsFromUnitDirectly0| unitFormp)
    (|sayKeyedMsg| 'S2IZ0041 (list unitForm)))))))))
```

———-

46.1.4 defun reportOpsFromLisplib0

[reportOpsFromLisplib1 p809]
[reportOpsFromLisplib p810]
[$useEditorForShowOutput p786]

— **defun reportOpsFromLisplib0** —

```
(defun |reportOpsFromLisplib0| (unitForm u)
 (declare (special |$useEditorForShowOutput|))
  (if |$useEditorForShowOutput|
   (|reportOpsFromLisplib1| unitForm u)
   (|reportOpsFromLisplib| unitForm u)))
```

———-

46.1.5 defun reportOpsFromLisplib1

[pathname p1036]
[erase p**??**]
[defiostream p972]
[sayShowWarning p818]
[reportOpsFromLisplib p810]
[shut p972]
[editFile p541]
[$sayBrightlyStream p**??**]
[$erase p**??**]

— **defun reportOpsFromLisplib1** —

```
(defun |reportOpsFromLisplib1| (unitForm u)
 (let (|$sayBrightlyStream| showFile)
 (declare (special |$sayBrightlyStream| $erase))
```

```
  (setq showFile (|pathname| (list 'show 'listing 'a)))
  ($erase showFile)
  (setq |$sayBrightlyStream|
   (defiostream `((file ,showFile) (mode . output)) 255 0))
  (|sayShowWarning|)
  (|reportOpsFromLisplib| unitForm u)
  (shut |$sayBrightlyStream|)
  (|editFile| showFile)))
```

———-

46.1.6 defun reportOpsFromLisplib

[constructor? p**??**]
[sayKeyedMsg p327]
[getConstructorSignature p**??**]
[kdr p**??**]
[getdatabase p1001]
[eqsubstlist p**??**]
[nreverse0 p**??**]
[sayBrightly p**??**]
[concat p1041]
[bright p**??**]
[form2StringWithWhere p**??**]
[isExposedConstructor p812]
[strconc p**??**]
[namestring p1034]
[selectOptionLC p475]
[dc1 p**??**]
[centerAndHighlight p**??**]
[specialChar p970]
[remdup p**??**]
[msort p**??**]
[form2String p**??**]
[say2PerLine p**??**]
[formatAttribute p**??**]
[displayOperationsFromLisplib p812]
[$linelength p765]
[$showOptions p**??**]
[$options p**??**]
[$FormalMapVariableList p**??**]

— defun reportOpsFromLisplib —

```
(defun |reportOpsFromLisplib| (op u)
```

```
(let (fn s typ nArgs argList functorForm argml tmp1 functorFormWithDecl
      verb sourceFile opt attList)
(declare (special $linelength |$showOptions| |$options|
                  |$FormalMapVariableList|))
 (if (null (setq fn (|constructor?| op)))
  (|sayKeyedMsg| 'S2IZ0054 (list u))
  (progn
   (setq argml (when (setq s (|getConstructorSignature| op)) (kdr s)))
   (setq typ (getdatabase op 'constructorkind))
   (setq nArgs (|#| argml))
   (setq argList (kdr (getdatabase op 'constructorform)))
   (setq functorForm (cons op argList))
   (setq argml (eqsubstlist argList |$FormalMapVariableList| argml))
   (mapcar #'(lambda (a m) (push (list '|:| a m) tmp1)) argList argml)
   (setq functorFormWithDecl (cons op (nreverse0 tmp1)))
   (|sayBrightly|
    (|concat| (|bright| (|form2StringWithWhere| functorFormWithDecl))
              " is a" (|bright| typ) "constructor"))
   (|sayBrightly|
    (cons " Abbreviation for"
     (append (|bright| op) (cons "is" (|bright| fn)))))
   (if (|isExposedConstructor| op)
    (setq verb "is")
    (setq verb "is not"))
   (|sayBrightly|
    (cons " This constructor"
     (append (|bright| verb) (list "exposed in this frame."))))
   (setq sourceFile (getdatabase op 'sourcefile))
   (|sayBrightly|
    (cons " Issue"
     (append (|bright| (strconc ")edit " (|namestring| sourceFile)))
             (cons "to see algebra source code for"
              (append (|bright| fn) (list '|%l|))))))
   (dolist (item |$options|)
    (setq opt (|selectOptionLC| (car item) |$showOptions| '|optionError|))
    (cond
     ((eq opt '|layout|) (|dc1| fn))
     ((eq opt '|views|)
      (|sayBrightly|
       (cons "To get" (append (|bright| "views")
         (list "you must give parameters of constructor")))))
     ((eq opt '|attributes|)
      (|centerAndHighlight| "Attributes" $linelength (|specialChar| '|hbar|))
      (|sayBrightly| "")
      (setq attList
       (remdup
        (msort
         (mapcar #'(lambda (x) (caar x))
          (reverse (getdatabase op 'attributes))))))
      (if (null attList)
```

```
        (|sayBrightly|
         (|concat| '|%b| (|form2String| functorForm)
                   '|%d| '|has no attributes.| '|%l|))
         (|say2PerLine| (mapcar #'|formatAttribute| attList))))
       ((eq opt '|operations|)
        (|displayOperationsFromLisplib| functorForm))))))))
```

———-

46.1.7 defun isExposedConstructor

[getalist p??]
[$localExposureData p688]
[$globalExposureGroupAlist p662]

— **defun isExposedConstructor** —

```
(defun |isExposedConstructor| (name)
 (let (x found)
 (declare (special |$globalExposureGroupAlist| |$localExposureData|))
  (cond
    ((member name '(|Union| |Record| |Mapping|)) t)
    ((member name (elt |$localExposureData| 2)) nil)
    ((member name (elt |$localExposureData| 1)) t)
    (t
     (loop for g in (elt |$localExposureData| 0)
      when (not found)
      do
        (setq x (getalist |$globalExposureGroupAlist| g))
        (when (and x (getalist x name)) (setq found t)))
     found))))
```

———-

46.1.8 defun displayOperationsFromLisplib

[getdatabase p1001]
[centerAndHighlight p??]
[specialChar p970]
[reportOpsFromUnitDirectly p813]
[remdup p??]
[msort p??]
[eqsubstlist p??]
[formatOperationAlistEntry p??]

[say2PerLine p??]
[$FormalMapVariableList p??]
[$linelength p765]

— **defun displayOperationsFromLisplib** —

```
(defun |displayOperationsFromLisplib| (form)
 (let (name argl kind opList opl ops)
 (declare (special |$FormalMapVariableList| $linelength))
  (setq name (car form))
  (setq argl (cdr form))
  (setq kind (getdatabase name 'constructorkind))
  (|centerAndHighlight| "Operations" $linelength (|specialChar| '|hbar|))
  (setq opList (getdatabase name 'operationalist))
  (if (null opList)
   (|reportOpsFromUnitDirectly| form)
   (progn
    (setq opl
     (remdup (msort (eqsubstlist argl |$FormalMapVariableList| opList))))
    (setq ops nil)
    (dolist (x opl)
     (setq ops (append ops (|formatOperationAlistEntry| x))))
    (|say2PerLine| ops)))))
```

———-

46.1.9 defun reportOpsFromUnitDirectly0

[reportOpsFromUnitDirectly1 p817]
[reportOpsFromUnitDirectly p813]
[$useEditorForShowOutput p786]

— **defun reportOpsFromUnitDirectly0** —

```
(defun |reportOpsFromUnitDirectly0| (D)
 (declare (special |$useEditorForShowOutput|))
  (if |$useEditorForShowOutput|
   (|reportOpsFromUnitDirectly1| D)
   (|reportOpsFromUnitDirectly| D)))
```

———-

46.1.10 defun reportOpsFromUnitDirectly

[member p1042]
[qcar p??]

[evalDomain p903]
[getdatabase p1001]
[sayBrightly p??]
[concat p1041]
[formatOpType p??]
[isExposedConstructor p812]
[bright p??]
[sayBrightly p??]
[strconc p??]
[namestring p1034]
[selectOptionLC p475]
[centerAndHighlight p??]
[specialChar p970]
[remdup p??]
[msort p??]
[formatAttribute p??]
[centerAndHighlight p??]
[getl p??]
[systemErrorHere p??]
[nreverse0 p??]
[getOplistForConstructorForm p816]
[say2PerLine p??]
[formatOperation p??]
[$commentedOps p??]
[$CategoryFrame p??]
[$linelength p765]
[$options p??]
[$showOptions p??]

— defun reportOpsFromUnitDirectly —

```
(defun |reportOpsFromUnitDirectly| (unitForm)
 (let (|$commentedOps| isRecordOrUnion unit top kind abb sourceFile verb opt
       attList constructorFunction tmp1 funlist a sigList tmp2)
 (declare (special |$commentedOps| |$CategoryFrame| $linelength |$options|
                   |$showOptions|))
  (setq isRecordOrUnion
   (and (consp unitForm)
        (progn (setq a (qcar unitForm)) t)
        (|member| a '(|Record| |Union|))))
  (setq unit (|evalDomain| unitForm))
  (setq top (car unitForm))
  (setq kind (getdatabase top 'constructorkind))
  (|sayBrightly|
   (|concat| '|%b| (|formatOpType| unitForm) '|%d|
             "is a" '|%b| kind '|%d| "constructor."))
  (unless isRecordOrUnion
```

```
     (setq abb (getdatabase top 'abbreviation))
     (setq sourceFile (getdatabase top 'sourcefile))
     (|sayBrightly|
      (cons " Abbreviation for"
       (append (|bright| top) (cons "is" (|bright| abb)))))
     (if (|isExposedConstructor| top)
      (setq verb "is")
      (setq verb "is not"))
     (|sayBrightly|
      (cons " This constructor"
       (append (|bright| verb) (list "exposed in this frame." ))))
     (|sayBrightly|
      (cons " Issue"
       (append (|bright| (strconc ")edit " (|namestring| sourceFile)))
               (cons "to see algebra source code for"
                (append (|bright| abb) (list '|%l|)))))))
  (dolist (item |$options|)
   (setq opt (|selectOptionLC| (car item) |$showOptions| '|optionError|))
   (cond
    ((eq opt '|attributes|)
     (|centerAndHighlight| "Attributes" $linelength (|specialChar| '|hbar|))
     (if isRecordOrUnion
       (|sayBrightly| "   Records and Unions have no attributes.")
       (progn
        (|sayBrightly| "")
        (setq attList
         (remdup
          (msort
           (mapcar #'(lambda (unit2) (car unit2)) (reverse (elt unit 2))))))
        (|say2PerLine|
         (mapcar #'|formatAttribute| attList))
        nil)))
    ((eq opt '|operations|)
     (setq |$commentedOps| 0)
;      --new form is (<op> <signature> <slotNumber> <condition> <kind>)
     (|centerAndHighlight| "Operations" $linelength (|specialChar| '|hbar|))
     (|sayBrightly| "")
     (cond
      (isRecordOrUnion
       (setq constructorFunction (getl top '|makeFunctionList|))
       (unless constructorFunction
        (|systemErrorHere| "reportOpsFromUnitDirectly"))
       (setq tmp1
        (funcall constructorFunction '$ unitForm |$CategoryFrame|))
       (setq funlist (car tmp1))
       (setq sigList
        (remdup
         (msort
          (dolist (fun funlist (nreverse0 tmp2))
           (push `(((,(caar fun) ,(cadar fun)) t (,(caddar fun) 0 1)))
```

```
            tmp2))))))
      (t
       (setq sigList
        (remdup (msort (|getOplistForConstructorForm| unitForm))))))
     (|say2PerLine|
      (mapcar #'(lambda (x) (|formatOperation| x unit)) sigList))
     (unless (= |$commentedOps| 0)
      (|sayBrightly|
       (list "Functions that are not yet implemented are preceded by"
             (|bright| "--"))))
     (|sayBrightly| "")))
  nil))
```

———-

46.1.11 defun getOplistForConstructorForm

The new form is an op-Alist which has entries

```
(<op> . signature-Alist)
```

where signature-Alist has entries

```
(<signature> . item)
```

where item has form (¡slotNumber¿ ¡condition¿ ¡kind¿)

```
(<slotNumber> <condition> <kind>)
```

where ¡kind¿ = ELT — CONST — Subsumed — (XLAM..) ..

```
<kind> =  ELT | CONST | Subsumed | (XLAM..) ..
```

— defun getOplistForConstructorForm —

```
(defun |getOplistForConstructorForm| (form)
 (let (argl pairlis opAlist op signatureAlist result)
 (declare (special |$FormalMapVariableList|))
  (setq op (car form))
  (setq argl (cdr form))
  (setq pairlis
   (loop for fv in |$FormalMapVariableList|
         for arg in argl
    collect (cons fv arg)))
  (setq opAlist (|getOperationAlistFromLisplib| op))
```

```
  (loop for item in opAlist do
    (setq op (car item))
    (setq signatureAlist (cdr item))
    (setq result
     (append result
      (|getOplistWithUniqueSignatures| op pairlis signatureAlist))))
  result))
```

———-

46.1.12 defun getOplistWithUniqueSignatures

— **defun getOplistWithUniqueSignatures** —

```
(defun |getOplistWithUniqueSignatures| (op pairlis signatureAlist)
 (let (sig slotNumber pred kind alist)
  (loop for item in signatureAlist
   when (not (eq (fourth item) '|Subsumed|))
   do
     (setq sig (first item))
     (setq slotNumber (second item))
     (setq pred (third item))
     (setq kind (fourth item))
     (setq alist
      (|insertAlist|
       (sublis pairlis (list op sig))
       (sublis pairlis (list pred (list kind  nil slotNumber)))
       alist)))
  alist))
```

———-

46.1.13 defun reportOpsFromUnitDirectly1

[pathname p1036]
[erase p**??**]
[defiostream p972]
[sayShowWarning p818]
[reportOpsFromUnitDirectly p813]
[shut p972]
[editFile p541]
[$sayBrightlyStream p**??**]
[$erase p**??**]

— defun reportOpsFromUnitDirectly1 —

```
(defun |reportOpsFromUnitDirectly1| (D)
 (let (|$sayBrightlyStream| showFile)
 (declare (special |$sayBrightlyStream| $erase))
  (setq showFile (|pathname| (list 'show 'listing 'a)))
  ($erase showFile)
  (setq |$sayBrightlyStream|
   (defiostream `((file ,showFile) (mode . output)) 255 0))
  (|sayShowWarning|)
  (|reportOpsFromUnitDirectly| D)
  (shut |$sayBrightlyStream|)
  (|editFile| showFile)))
```

———-

46.1.14 defun sayShowWarning

[sayBrightly p??]

— defun sayShowWarning —

```
(defun |sayShowWarning| ()
 (|sayBrightly|
   "Warning: this is a temporary file and will be deleted the next")
 (|sayBrightly|
   "         time you use )show. Rename it and FILE if you wish to")
 (|sayBrightly| "         save the contents.")
 (|sayBrightly| ""))
```

———-

Chapter 47

)spool help page Command

47.1 spool help page man page

— spool.help —

```
====================================================================
A.23.  )spool
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )spool [fileName]
  - )spool

Command Description:

This command is used to save (spool) all AXIOM input and output into a file,
called a spool file. You can only have one spool file active at a time. To
start spool, issue this command with a filename. For example,

)spool integrate.out

To stop spooling, issue )spool with no filename.

If the filename is qualified with a directory, then the output will be placed
in that directory. If no directory information is given, the spool file will
be placed in the current directory. The current directory is the directory
from which you started AXIOM or is the directory you specified using the )cd
command.
```

Also See:
o)cd

1

[1] "cd" (?? p ??)

Chapter 48

)summary help page Command

48.1 summary help page man page

— summary.help —

```
)credits      : list the people who have contributed to Axiom

)help <command> gives more information
)quit         : exit AXIOM

)abbreviation : query, set and remove abbreviations for constructors
)cd           : set working directory
)clear        : remove declarations, definitions or values
)close        : throw away an interpreter client and workspace
)compile      : invoke constructor compiler
)copyright    : show copyright and trademark information
)describe     : show database information for a category, domain, or package
)display      : display Library operations and objects in your workspace
)edit         : edit a file
)frame        : manage interpreter workspaces
)history      : manage aspects of interactive session
)library      : introduce new constructors
)lisp         : evaluate a LISP expression
)read         : execute AXIOM commands from a file
)savesystem   : save LISP image to a file
)set          : view and set system variables
)show         : show constructor information
)spool        : log input and output to a file
)synonym      : define an abbreviation for system commands
)system       : issue shell commands
)trace        : trace execution of functions
)undo         : restore workspace to earlier state
```

```
)what         : search for various things by name
```

———-

48.1.1 defun summary

[obey p**??**]
[concat p1041]
[getenviron p31]

— **defun summary** —

```
(defun |summary| (l)
 (declare (ignore l))
 (obey (concat "cat " (getenviron "AXIOM") "/doc/spadhelp/summary.help")))
```

———-

Chapter 49

)synonym help page Command

49.1 synonym help page man page

— synonym.help —

```
====================================================================
A.24.  )synonym
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )synonym
  - )synonym synonym fullCommand
  - )what synonyms

Command Description:

This command is used to create short synonyms for system command expressions.
For example, the following synonyms might simplify commands you often use.

)synonym save         history )save
)synonym restore      history )restore
)synonym mail         system mail
)synonym ls           system ls
)synonym fortran      set output fortran

Once defined, synonyms can be used in place of the longer command
expressions. Thus

)fortran on
```

```
is the same as the longer

)set fortran output on

To list all defined synonyms, issue either of

)synonyms
)what synonyms

To list, say, all synonyms that contain the substring ‘‘ap’’, issue

)what synonyms ap

Also See:
o )set
o )what
```

————-

[1]

49.1.1 defun The)synonym command

[synonymSpad2Cmd p824]

— **defun synonym** —

```
(defun |synonym| (&rest ignore)
 (declare (ignore ignore))
 (|synonymSpad2Cmd|))
```

————-

49.1.2 defun The)synonym command implementation

[getSystemCommandLine p825]
[printSynonyms p470]
[processSynonymLine p827]
[putalist p**??**]
[terminateSystemCommand p448]
[$CommandSynonymAlist p474]

— **defun synonymSpad2Cmd** —

[1] “set” (45.36.1 p 799) “what” (53.1.2 p 929)

```
(defun |synonymSpad2Cmd| ()
 (let (line pair)
 (declare (special |$CommandSynonymAlist|))
  (setq line (|getSystemCommandLine|))
  (if (string= line "")
   (|printSynonyms| nil)
   (progn
    (setq pair (|processSynonymLine| line))
    (if |$CommandSynonymAlist|
     (putalist |$CommandSynonymAlist| (car pair) (cdr pair))
     (setq |$CommandSynonymAlist| (cons pair nil)))))
  (|terminateSystemCommand|)))
```

———-

49.1.3 defun Return a sublist of applicable synonyms

The argument is a list of synonyms, and this returns a sublist of applicable synonyms at the current user level. [string2id-n p**??**]
[selectOptionLC p475]
[commandsForUserLevel p444]
[$systemCommands p439]
[$UserLevel p798]

— **defun synonymsForUserLevel** —

```
(defun |synonymsForUserLevel| (arg)
 (let (cmd nl)
 (declare (special |$systemCommands| |$UserLevel|))
  (if (eq |$UserLevel| '|development|)
   arg
   (dolist (syn (reverse arg))
    (setq cmd (string2id-n (cdr syn) 1))
    (when (|selectOptionLC| cmd (|commandsForUserLevel| |$systemCommands|) nil)
     (push syn nl))))
   nl))
```

———-

49.1.4 defun Get the system command from the input line

[strpos p1039]
[substring p**??**]
[$currentLine p**??**]

— **defun getSystemCommandLine** —

```
(defun |getSystemCommandLine| ()
 (let (p line)
 (declare (special |$currentLine|))
  (setq p (strpos ")" |$currentLine| 0 nil))
  (if p
   (setq line (substring |$currentLine| p nil))
   (setq line |$currentLine|))
 (string-left-trim '(#\space) line)))
```

———-

49.1.5 defun Remove system keyword

[dropLeadingBlanks p**??**]
[maxindex p**??**]

— **defun processSynonymLine,removeKeyFromLine** —

```
(defun |processSynonymLine,removeKeyFromLine| (line)
 (prog (mx)
  (return
   (seq
    (setq line (|dropLeadingBlanks| line))
    (setq mx (maxindex line))
    (exit
     (do ((i 0 (1+ i)))
         ((> i mx) nil)
       (seq
        (exit
         (if (char= (elt line i) #\space)
          (exit
           (return
            (do ((j (1+ i) (1+ j)))
                ((> j mx) nil)
              (seq
               (exit
                (if (char\= (elt line j) #\space)
                 (exit
                  (return
                   (substring line j nil)))))))))))))))))
```

———-

49.1.6 defun processSynonymLine

[processSynonymLine,removeKeyFromLine p826]

— **defun processSynonymLine** —

```
(defun |processSynonymLine| (line)
 (cons
  (string2id-n line 1)
  (|processSynonymLine,removeKeyFromLine| line)))
```

————-

This command is in the list of `$noParseCommands` 19.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 19.2.1

Chapter 50

)system help page Command

50.1 system help page man page

— system.help —

```
=====================================================================
A.25.  )system
=====================================================================

User Level Required:  interpreter

Command Syntax:

  - )system cmdExpression

Command Description:

This command may be used to issue commands to the operating system while
remaining in AXIOM. The cmdExpression is passed to the operating system for
execution.

To get an operating system shell, issue, for example, )system sh. When you
enter the key combination, Ctrl-D (pressing and holding the Ctrl key and then
pressing the D key) the shell will terminate and you will return to AXIOM. We
do not recommend this way of creating a shell because Lisp may field some
interrupts instead of the shell. If possible, use a shell running in another
window.

If you execute programs that misbehave you may not be able to return to
AXIOM. If this happens, you may have no other choice than to restart AXIOM
and restore the environment via )history )restore, if possible.
```

```
Also See:
o )boot
o )fin
o )lisp
o )pquit
o )quit
```

———-

1

This command is in the list of `$noParseCommands` 19.1.3 which means that its arguments are passed verbatim. This will eventually result in a call to the function `handleNoParseCommands` 19.2.1

[1] "boot" (5.1.8 p 25) "fin" (32.1.1 p 544) "lisp" (?? p ??) "pquit" (41.2.1 p 630) "quit" (42.2.1 p 634)

Chapter 51

)trace help page Command

51.1 trace help page man page

— trace.help —

```
========================================================================
A.26.  )trace
========================================================================

User Level Required:  interpreter

Command Syntax:

  - )trace
  - )trace )off

  - )trace function [options]
  - )trace constructor [options]
  - )trace domainOrPackage [options]

where options can be one or more of

  - )after S-expression
  - )before S-expression
  - )break after
  - )break before
  - )cond S-expression
  - )count
  - )count n
  - )depth n
  - )local op1 [... opN]
  - )nonquietly
```

```
  - )nt
  - )off
  - )only listOfDataToDisplay
  - )ops
  - )ops op1 [... opN ]
  - )restore
  - )stats
  - )stats reset
  - )timer
  - )varbreak
  - )varbreak var1 [... varN ]
  - )vars
  - )vars var1 [... varN ]
  - )within executingFunction

Command Description:

This command is used to trace the execution of functions that make up the
AXIOM system, functions defined by users, and functions from the system
library. Almost all options are available for each type of function but
exceptions will be noted below.

To list all functions, constructors, domains and packages that are traced,
simply issue

)trace

To untrace everything that is traced, issue

)trace )off

When a function is traced, the default system action is to display the
arguments to the function and the return value when the function is exited.
Note that if a function is left via an action such as a THROW, no return
value will be displayed. Also, optimization of tail recursion may decrease
the number of times a function is actually invoked and so may cause less
trace information to be displayed. Other information can be displayed or
collected when a function is traced and this is controlled by the various
options. Most options will be of interest only to AXIOM system developers. If
a domain or package is traced, the default action is to trace all functions
exported.

Individual interpreter, lisp or boot functions can be traced by listing their
names after )trace. Any options that are present must follow the functions to
be traced.

)trace f

traces the function f. To untrace f, issue
```

```
)trace f )off

Note that if a function name contains a special character, it will be
necessary to escape the character with an underscore

)trace _/D_,1

To trace all domains or packages that are or will be created from a
particular constructor, give the constructor name or abbreviation after
)trace.

)trace MATRIX
)trace List Integer

The first command traces all domains currently instantiated with Matrix. If
additional domains are instantiated with this constructor (for example, if
you have used Matrix(Integer) and Matrix(Float)), they will be automatically
traced. The second command traces List(Integer). It is possible to trace
individual functions in a domain or package. See the )ops option below.

The following are the general options for the )trace command.

  )break after
    causes a Lisp break loop to be entered after exiting the traced function.

  )break before
    causes a Lisp break loop to be entered before entering the traced
    function.

  )break
    is the same as )break before.

  )count
    causes the system to keep a count of the number of times the traced
    function is entered. The total can be displayed with )trace )stats and
    cleared with )trace )stats reset.

  )count n
    causes information about the traced function to be displayed for the
    first n executions. After the nth execution, the function is untraced.

  )depth n
    causes trace information to be shown for only n levels of recursion of
    the traced function. The command

    )trace fib )depth 10

    will cause the display of only 10 levels of trace information for the
    recursive execution of a user function fib.
```

```
)math
  causes the function arguments and return value to be displayed in the
  AXIOM monospace two-dimensional math format.

)nonquietly
  causes the display of additional messages when a function is traced.

)nt
  This suppresses all normal trace information. This option is useful if
  the )count or )timer options are used and you are interested in the
  statistics but not the function calling information.

)off
  causes untracing of all or specific functions. Without an argument, all
  functions, constructors, domains and packages are untraced. Otherwise,
  the given functions and other objects are untraced. To immediately
  retrace the untraced functions, issue )trace )restore.

)only listOfDataToDisplay
  causes only specific trace information to be shown. The items are listed
  by using the following abbreviations:

  a        display all arguments
  v        display return value
  1        display first argument
  2        display second argument
  15       display the 15th argument, and so on

)restore
  causes the last untraced functions to be retraced. If additional options
  are present, they are added to those previously in effect.

)stats
  causes the display of statistics collected by the use of the )count and
  )timer options.

)stats reset
  resets to 0 the statistics collected by the use of the )count and )timer
  options.

)timer
  causes the system to keep a count of execution times for the traced
  function. The total can be displayed with )trace )stats and cleared with
  )trace )stats reset.

)varbreak var1 [... varN]
  causes a Lisp break loop to be entered after the assignment to any of the
  listed variables in the traced function.

)vars
```

```
    causes the display of the value of any variable after it is assigned in
    the traced function. Note that library code must have been compiled (see
    description of command )compile ) using the )vartrace option in order to
    support this option.

  )vars var1 [... varN]
    causes the display of the value of any of the specified variables after
    they are assigned in the traced function. Note that library code must
    have been compiled (see description of command )compile ) using the
    )vartrace option in order to support this option.

  )within executingFunction
    causes the display of trace information only if the traced function is
    called when the given executingFunction is running.

The following are the options for tracing constructors, domains and packages.

  )local [op1 [... opN] ]
    causes local functions of the constructor to be traced. Note that to
    untrace an individual local function, you must use the fully qualified
    internal name, using the escape character _ before the semicolon.

    )trace FRAC )local
    )trace FRAC_;cancelGcd )off

  )ops op1 [... opN]
    By default, all operations from a domain or package are traced when the
    domain or package is traced. This option allows you to specify that only
    particular operations should be traced. The command

    )trace Integer )ops min max _+ _-

    traces four operations from the domain Integer. Since + and - are special
    characters, it is necessary to escape them with an underscore.

Also See:
o )boot
o )lisp
o )ltrace
```

———————

[1]

51.1.1 The trace global variables

This decides when to give trace and untrace messages.

[1] "boot" (5.1.8 p 25) "lisp" (?? p ??) "ltrace" (40.1.1 p 628)

51.1.2 defvar $traceNoisely

— **initvars** —

```
(defvar |$traceNoisely| nil)
```

———-

51.1.3 defvar $reportSpadTrace

This reports the traced functions

— **initvars** —

```
(defvar |$reportSpadTrace| nil)
```

———-

51.1.4 defvar $optionAlist

— **initvars** —

```
(defvar |$optionAlist| nil)
```

———-

51.1.5 defvar $tracedMapSignatures

— **initvars** —

```
(defvar |$tracedMapSignatures| nil)
```

———-

51.1.6 defvar $traceOptionList

— **initvars** —

```
(defvar |$traceOptionList|
 '(|after| |before| |break| |cond| |count| |depth| |local| |mathprint|
   |nonquietly| |nt| |of| |only| |ops| |restore| |timer| |varbreak|
   |vars| |within|))
```

———-

51.1.7 defun trace

[traceSpad2Cmd p837]

— **defun trace** —

```
(defun |trace| (l)
 (|traceSpad2Cmd| l))
```

———-

51.1.8 defun traceSpad2Cmd

[qcar p??]
[qcdr p??]
[getMapSubNames p859]
[trace1 p838]
[augmentTraceNames p862]
[traceReply p889]
[$mapSubNameAlist p??]

— **defun traceSpad2Cmd** —

```
(defun |traceSpad2Cmd| (l)
 (let (tmp1 l1)
 (declare (special |$mapSubNameAlist|))
  (cond
   ((and (consp l)
         (eq (qcar l) '|Tuple|)
         (progn
          (setq tmp1 (qcdr l))
          (and (consp tmp1)
               (eq (qcdr tmp1) nil)
               (progn
                (setq l1 (qcar tmp1))
                t))))
     (setq l l1)))
   (setq |$mapSubNameAlist| (|getMapSubNames| l))
```

```
(|trace1| (|augmentTraceNames| l))
(|traceReply|)))
```

———-

51.1.9 defun trace1

[hasOption p447]
[throwKeyedMsg p??]
[unabbrev p??]
[isFunctor p??]
[getTraceOption p844]
[untraceDomainLocalOps p870]
[qslessp p1053]
[poundsign p??]
[untrace p852]
[centerAndHighlight p??]
[ptimers p849]
[say p??]
[pcounters p850]
[selectOptionLC p475]
[resetSpacers p848]
[resetTimers p848]
[resetCounters p848]
[qcar p??]
[qcdr p??]
[vecp p??]
[sayKeyedMsg p327]
[devaluate p??]
[lassoc p??]
[trace1 p838]
[delete p??]
[?t p892]
[seq p??]
[exit p??]
[transTraceItem p853]
[addassoc p??]
[getTraceOptions p842]
[/trace,0 p??]
[saveMapSig p843]
[$traceNoisely p836]
[$options p??]
[$lastUntraced p??]
[$optionAlist p836]

— defun trace1 —

```
(defun |trace1| (arg)
 (prog (|$traceNoisely| constructor ops lops temp1 opt a
        oldl newoptions domain tracelist optionlist domainlist
        oplist y varlist argument)
  (declare (special |$traceNoisely| |$options| |$lastUntraced|
     |$optionAlist|))
  (return
   (seq
    (progn
     (setq |$traceNoisely| nil)
     (cond
      ((|hasOption| |$options| '|nonquietly|)
        (setq |$traceNoisely| t)))
     (cond
      ((|hasOption| |$options| '|off|)
       (cond
        ((or (setq ops (|hasOption| |$options| 'ops))
             (setq lops (|hasOption| |$options| '|local|)))
          (cond
           ((null arg) (|throwKeyedMsg| 's2it0019 nil))
           (t
             (setq constructor
              (|unabbrev|
               (cond
                ((atom arg) arg)
                ((null (cdr arg))
                  (cond
                   ((atom (car arg)) (car arg))
                   (t (car (car arg)))))
                (t nil))))
             (cond
              ((null (|isFunctor| constructor))
                (|throwKeyedMsg| 's2it0020 nil))
              (t
                (cond (ops (setq ops (|getTraceOption| ops)) nil))
                (cond
                 (lops
                  (setq lops (cdr (|getTraceOption| lops)))
                  (|untraceDomainLocalOps|))
                 (t nil)))))))
        ((and (qslessp 1 (|#| |$options|))
              (null (|hasOption| |$options| '|nonquietly|)))
          (|throwKeyedMsg| 's2it0021 nil))
        (t (|untrace| arg))))
      ((|hasOption| |$options| '|stats|)
        (cond
         ((qslessp 1 (|#| |$options|))
```

```
     (|throwKeyedMsg| 's2it0001 (cons ")trace ... )stats" nil)))
    (t
      (setq temp1 (car |$options|))
      (setq opt (cdr temp1))
      (cond
       ((null opt)
         (|centerAndHighlight| "Traced function execution times" 78 '-)
         (|ptimers|)
         (say " ")
         (|centerAndHighlight| "Traced function execution counts" 78 '-)
         (|pcounters|))
       (t
         (|selectOptionLC| (car opt) '(|reset|) '|optionError|)
         (|resetSpacers|)
         (|resetTimers|)
         (|resetCounters|)
         (|throwKeyedMsg| 's2it0002 nil))))))
((setq a (|hasOption| |$options| '|restore|))
  (unless (setq oldl |$lastUntraced|)
   (setq newoptions (|delete| a |$options|))
   (if (null arg)
    (|trace1| oldl)
    (progn
     (dolist (x arg)
      (if (and (consp x)
               (progn
                (setq domain (qcar x))
                (setq oplist (qcdr x))
                t)
               (vecp domain))
       (|sayKeyedMsg| 's2it0003 (cons (|devaluate| domain) nil))
       (progn
         (setq |$options| (append newoptions (lassoc x |$optionAlist|)))
         (|trace1| (list x)))))))))
((null arg) nil)
((and (consp arg) (eq (qcdr arg) nil) (eq (qcar arg) '?)) (|?t|))
(t
 (setq tracelist
  (or
   (prog (t1)
    (setq t1 nil)
    (return
     (do ((t2 arg (cdr t2)) (x nil))
         ((or (atom t2)
              (progn (setq x (car t2)) nil))
          (nreverse0 t1))
      (seq
       (exit
        (setq t1 (cons (|transTraceItem| x) t1)))))))
   (return nil)))
```

```
(do ((t3 tracelist (cdr t3)) (x nil))
    ((or (atom t3) (progn (setq x (car t3)) nil)) nil)
  (seq
   (exit
    (setq |$optionAlist| (addassoc x |$options| |$optionAlist|)))))
(setq optionlist (|getTraceOptions| |$options|))
(setq argument
 (cond
  ((setq domainlist (lassoc '|of| optionlist))
    (cond
     ((lassoc 'ops optionlist)
       (|throwKeyedMsg| 's2it0004 nil))
     (t
       (setq oplist
        (cond
         (tracelist (list (cons 'ops tracelist)))
         (t nil)))
       (setq varlist
        (cond
         ((setq y (lassoc '|vars| optionlist))
           (list (cons '|vars| y)))
         (t nil)))
       (append domainlist (append oplist varlist)))))
  (optionlist (append tracelist optionlist))
  (t tracelist)))
(|/TRACE,0|
 (prog (t4)
  (setq t4 nil)
  (return
   (do ((t5 argument (cdr t5)) (|funName| nil))
       ((or (atom t5)
            (progn (setq |funName| (car t5)) nil))
         (nreverse0 t4))
    (seq
     (exit
      (setq t4 (cons |funName| t4))))))))
(|saveMapSig|
 (prog (t6)
  (setq t6 nil)
  (return
   (do ((t7 argument (cdr t7)) (|funName| nil))
       ((or (atom t7)
            (progn (setq |funName| (car t7)) nil))
         (nreverse0 t6))
    (seq
     (exit
      (setq t6 (cons |funName| t6)))))))))))))))
```

———-

51.1.10 defun getTraceOptions

[throwKeyedMsg p??]
[throwListOfKeyedMsgs p??]
[poundsign p??]
[seq p??]
[exit p??]
[getTraceOption p844]
[$traceErrorStack p??]

— **defun getTraceOptions** —

```
(defun |getTraceOptions| (|options|)
 (prog (|$traceErrorStack| optionlist temp1 key |parms|)
  (declare (special |$traceErrorStack|))
  (return
   (seq
    (progn
     (setq |$traceErrorStack| nil)
     (setq optionlist
      (prog (t0)
       (setq t0 nil)
       (return
        (do ((t1 |options| (cdr t1)) (x nil))
            ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
         (seq
          (exit
           (setq t0 (cons (|getTraceOption| x) t0))))))))
     (cond
      (|$traceErrorStack|
       (cond
        ((null (cdr |$traceErrorStack|))
          (setq temp1 (car |$traceErrorStack|))
          (setq key (car temp1))
          (setq |parms| (cadr temp1))
          (|throwKeyedMsg| key (cons "" |parms|)))
        (t
          (|throwListOfKeyedMsgs| 's2it0017
           (cons (|#| |$traceErrorStack|) nil)
           (nreverse |$traceErrorStack|)))))
      (t optionlist)))))))
```

———-

51.1.11 defun saveMapSig

[rassoc p??]
[addassoc p??]
[getMapSig p843]
[$tracedMapSignatures p836]
[$mapSubNameAlist p??]

— **defun saveMapSig** —

```
(defun |saveMapSig| (funnames)
 (let (map)
 (declare (special |$tracedMapSignatures| |$mapSubNameAlist|))
  (dolist (name funnames)
   (when (setq map (|rassoc| name |$mapSubNameAlist|))
     (setq |$tracedMapSignatures|
       (addassoc name (|getMapSig| map name) |$tracedMapSignatures|))))))
```

———-

51.1.12 defun getMapSig

[get p??]
[boot-equal p??]
[$InteractiveFrame p??]

— **defun getMapSig** —

```
(defun |getMapSig| (mapname subname)
 (let (lmms sig)
 (declare (special |$InteractiveFrame|))
  (when (setq lmms (|get| mapname '|localModemap| |$InteractiveFrame|))
    (do ((t0 lmms (cdr t0)) (|mm| nil) (t1 nil sig))
        ((or (atom t0) (progn (setq |mm| (car t0)) nil) t1) nil)
      (when (boot-equal (cadr |mm|) subname) (setq sig (cdar |mm|))))
    sig)))
```

———-

51.1.13 defun getTraceOption,hn

[seq p??]
[exit p??]
[isDomainOrPackage p865]

[stackTraceOptionError p851]
[domainToGenvar p851]

— **defun getTraceOption,hn** —

```
(defun |getTraceOption,hn| (x)
 (prog (g)
  (return
   (seq
    (if (and (atom x) (null (upper-case-p (elt (princ-to-string x) 0))))
     (exit
      (seq
       (if (|isDomainOrPackage| (eval x)) (exit x))
       (exit
        (|stackTraceOptionError|
         (cons 's2it0013 (cons (cons x nil) nil)))))))
    (if (setq g (|domainToGenvar| x)) (exit g))
    (exit
     (|stackTraceOptionError| (cons 's2it0013 (cons (cons x nil) nil))))))))
```

———-

51.1.14 defun getTraceOption

[seq p**??**]
[exit p**??**]
[selectOptionLC p475]
[identp p1040]
[stackTraceOptionError p851]
[concat p1041]
[object2String p**??**]
[transOnlyOption p850]
[qcdr p**??**]
[qcar p**??**]
[getTraceOption,hn p843]
[isListOfIdentifiersOrStrings p859]
[isListOfIdentifiers p858]
[throwKeyedMsg p**??**]
[$traceOptionList p836]

— **defun getTraceOption** —

```
(defun |getTraceOption| (arg)
 (prog (l |opts| key a |n|)
 (declare (special |$traceOptionList|))
  (return
```

```
(seq
 (progn
  (setq key (car arg))
  (setq l (cdr arg))
  (setq key
    (|selectOptionLC| key |$traceOptionList| '|traceOptionError|))
  (setq arg (cons key l))
  (cond
   ((member key '(|nonquietly| |timer| |nt|)) arg)
   ((eq key '|break|)
    (cond
     ((null l) (cons '|break| (cons '|before| nil)))
     (t
       (setq |opts|
        (prog (t0)
         (setq t0 nil)
         (return
          (do ((t1 l (cdr t1)) (y nil))
              ((or (atom t1)
                   (progn (setq y (car t1)) nil))
                 (nreverse0 t0))
           (seq
            (exit
             (setq t0
              (cons
               (|selectOptionLC| y '(|before| |after|) nil) t0))))))))
      (cond
       ((prog (t2)
         (setq t2 t)
         (return
          (do ((t3 nil (null t2)) (t4 |opts| (cdr t4)) (y nil))
              ((or t3 (atom t4) (progn (setq y (car t4)) nil)) t2)
             (seq
              (exit
               (setq t2 (and t2 (identp y))))))
          (cons '|break| |opts|)))
        (t
         (|stackTraceOptionError| (cons 's2it0008 (cons nil nil))))))))
   ((eq key '|restore|)
    (cond
     ((null l) arg)
     (t
      (|stackTraceOptionError|
       (cons 's2it0009
        (cons (cons (concat ")" (|object2String| key)) nil) nil))))))
   ((eq key '|only|) (cons '|only| (|transOnlyOption| l)))
   ((eq key '|within|)
     (cond
      ((and (consp l)
            (eq (qcdr l) nil)
```

```
          (progn (setq a (qcar l)) t)
          (identp a))
      arg)
     (t
      (|stackTraceOptionError|
       (cons 's2it0010 (cons (cons ")within" nil) nil))))))
((member key '(|cond| |before| |after|))
  (setq key
   (cond
    ((eq key '|cond|) '|when|)
    (t key)))
  (cond
   ((and (consp l)
         (eq (qcdr l) nil)
         (progn (setq a (qcar l)) t))
      (cons key l))
   (t
    (|stackTraceOptionError|
     (cons 's2it0011
      (cons
       (cons (concat ")"
        (|object2String| key)) nil) nil))))))
((eq key '|depth|)
  (cond
   ((and (consp l)
         (eq (qcdr l) nil)
         (progn (setq |n| (qcar l)) t)
         (integerp |n|))
     arg)
   (t
    (|stackTraceOptionError|
     (cons 's2it0012 (cons (cons ")depth" nil) nil))))))
((eq key '|count|)
  (cond
   ((or (null l)
        (and (consp l)
             (eq (qcdr l) nil)
             (progn (setq |n| (qcar l)) t)
             (integerp |n|)))
      arg)
   (t
    (|stackTraceOptionError|
      (cons 's2it0012 (cons (cons ")count" nil) nil))))))
((eq key '|of|)
  (cons '|of|
   (prog (t5)
    (setq t5 nil)
    (return
     (do ((t6 l (cdr t6)) (y nil))
         ((or (atom t6) (progn (setq y (car t6)) nil)) (nreverse0 t5))
```

```
          (seq
           (exit
            (setq t5 (cons (|getTraceOption,hn| y) t5)))))))))
      ((member key '(|local| ops |vars|))
        (cond
         ((or (null l)
              (and (consp l) (eq (qcdr l) nil) (eq (qcar l) '|all|)))
          (cons key '|all|))
         ((|isListOfIdentifiersOrStrings| l) arg)
         (t
           (|stackTraceOptionError|
            (cons 's2it0015
             (cons
              (cons (concat ")" (|object2String| key)) nil) nil))))))
      ((eq key '|varbreak|)
        (cond
         ((or (null l)
              (and (consp l) (eq (qcdr l) nil) (eq (qcar l) '|all|)))
          (cons '|varbreak| '|all|))
         ((|isListOfIdentifiers| l) arg)
         (t
           (|stackTraceOptionError|
            (cons 's2it0016
             (cons
              (cons (concat ")" (|object2String| key)) nil) nil))))))
      ((eq key '|mathprint|)
        (cond
         ((null l) arg)
         (t
           (|stackTraceOptionError|
            (cons 's2it0009
             (cons
              (cons (concat ")" (|object2String| key)) nil) nil))))))
      (key (|throwKeyedMsg| 's2it0005 (cons key nil)))))))))
```

———-

51.1.15 defun traceOptionError

[stackTraceOptionError p851]
[commandAmbiguityError p448]

— **defun traceOptionError** —

```
(defun |traceOptionError| (opt keys)
 (if (null keys)
  (|stackTraceOptionError| (cons 's2it0007 (cons (cons opt nil) nil)))
```

```
  (|commandAmbiguityError| '|trace option| opt keys)))
```

———-

51.1.16 defun resetTimers

[concat p1041]
[/timerlist p**??**]

— **defun resetTimers** —

```
(defun |resetTimers| ()
 (declare (special /timerlist))
 (dolist (timer /timerlist)
  (set (intern (concat timer ",TIMER")) 0)))
```

———-

51.1.17 defun resetSpacers

[concat p1041]
[/spacelist p**??**]

— **defun resetSpacers** —

```
(defun |resetSpacers| ()
 (declare (special /spacelist))
 (dolist (spacer /spacelist)
   (set (intern (concat spacer ",SPACE")) 0)))
```

———-

51.1.18 defun resetCounters

[concat p1041]
[/countlist p**??**]

— **defun resetCounters** —

```
(defun |resetCounters| ()
 (declare (special /countlist))
 (dolist (k /countlist)
  (set (intern (concat k ",COUNT")) 0)))
```

———-

51.1.19 defun ptimers

[sayBrightly p??]
[bright p??]
[quotient p??]
[concat p1041]
[float p??]
[/timerlist p??]

— **defun ptimers** —

```
(defun |ptimers| ()
  (declare (special /timerlist |$timerTicksPerSecond|))
  (if (null /timerlist)
   (|sayBrightly| "   no functions are timed")
   (dolist (timer /timerlist)
     (|sayBrightly|
      `("  " ,@(|bright| timer) |:| " "
         ,(quotient (eval (intern (concat timer ",TIMER")))
            (|float| |$timerTicksPerSecond|)) " sec.")))))
```

———-

51.1.20 defun pspacers

[sayBrightly p??]
[bright p??]
[concat p1041]
[/spacelist p??]

— **defun pspacers** —

```
(defun |pspacers| ()
 (declare (special /spacelist))
 (if (null /spacelist)
   (|sayBrightly| "   no functions have space monitored")
   (dolist (spacer /spacelist)
     (|sayBrightly|
      `("  " ,@(|bright| spacer) |: |
        ,(eval (intern (concat spacer ",SPACE"))) " bytes")))))
```

———-

51.1.21 defun pcounters

[sayBrightly p**??**]
[bright p**??**]
[concat p1041]
[/countlist p**??**]

— **defun pcounters** —

```
(defun |pcounters| ()
 (declare (special /countlist))
 (if (null /countlist)
   (|sayBrightly| "   no functions are being counted")
   (dolist (k /countlist)
     (|sayBrightly|
      `("  " ,@(|bright| k) |:| " " ,(eval (intern (concat k ",COUNT")))
        " times")))))
```

———-

51.1.22 defun transOnlyOption

[transOnlyOption p850]
[upcase p**??**]
[stackTraceOptionError p851]
[qcar p**??**]
[qcdr p**??**]

— **defun transOnlyOption** —

```
(defun |transOnlyOption| (arg)
 (let (y n)
  (when (and (consp arg) (progn (setq n (qcar arg)) (setq y (qcdr arg)) t))
    (cond
     ((integerp n) (cons n (|transOnlyOption| y)))
     ((member (setq n (upcase n)) '(v a c)) (cons n (|transOnlyOption| y)))
     (t
        (|stackTraceOptionError| (cons 's2it0006 (list (list n))))
        (|transOnlyOption| y))))))
```

———-

51.1.23 defun stackTraceOptionError

[$traceErrorStack p??]

— **defun stackTraceOptionError** —

```
(defun |stackTraceOptionError| (x)
 (declare (special |$traceErrorStack|))
 (push x |$traceErrorStack|)
 nil)
```

———-

51.1.24 defun removeOption

— **defun removeOption** —

```
(defun |removeOption| (op options)
 (let (opt t0)
  (do ((t1 options (cdr t1)) (optentry nil))
      ((or (atom t1)
           (progn (setq optentry (car t1)) nil)
           (progn (progn (setq opt (car optentry)) optentry) nil))
    (nreverse0 t0))
  (when (not (equal opt op)) (setq t0 (cons optentry t0))))))
```

———-

51.1.25 defun domainToGenvar

[unabbrevAndLoad p??]
[getdatabase p1001]
[opOf p??]
[genDomainTraceName p852]
[evalDomain p903]
[$doNotAddEmptyModeIfTrue p??]

— **defun domainToGenvar** —

```
(defun |domainToGenvar| (arg)
 (let (|$doNotAddEmptyModeIfTrue| y g)
 (declare (special |$doNotAddEmptyModeIfTrue|))
  (setq |$doNotAddEmptyModeIfTrue| t)
```

```
  (when
   (and (setq y (|unabbrevAndLoad| arg))
        (eq (getdatabase (|opOf| y) 'constructorkind) '|domain|))
     (setq g (|genDomainTraceName| y))
     (set g (|evalDomain| y))
     g)))
```

———-

51.1.26 defun genDomainTraceName

[lassoc p**??**]
[genvar p**??**]
[$domainTraceNameAssoc p**??**]

— **defun genDomainTraceName** —

```
(defun |genDomainTraceName| (y)
 (let (u g)
 (declare (special |$domainTraceNameAssoc|))
   (if (setq u (lassoc y |$domainTraceNameAssoc|))
    u
    (progn
     (setq g (genvar))
     (setq |$domainTraceNameAssoc| (cons (cons y g) |$domainTraceNameAssoc|))
     g))))
```

———-

51.1.27 defun untrace

[copy p**??**]
[transTraceItem p853]
[/untrace,0 p**??**]
[lassocSub p861]
[removeTracedMapSigs p854]
[$lastUntraced p**??**]
[$mapSubNameAlist p**??**]
[/tracenames p**??**]

— **defun untrace** —

```
(defun |untrace| (arg)
 (let (untracelist)
```

```
 (declare (special |$lastUntraced| /tracenames |$mapSubNameAlist|))
  (if arg
   (setq |$lastUntraced| arg)
   (setq |$lastUntraced| (copy /tracenames)))
  (setq untracelist
   (do ((t1 arg (cdr t1)) (x nil) (t0 nil))
       ((or (atom t1) (progn (setq x (car t1)) nil))
      (nreverse0 t0))
     (push (|transTraceItem| x) t0)))
  (|/UNTRACE,0|
   (do ((t3 untracelist (cdr t3)) (|funName| nil) (t2 nil))
       ((or (atom t3) (progn (setq |funName| (car t3)) nil))
     (nreverse0 t2))
    (push (|lassocSub| |funName| |$mapSubNameAlist|) t2)))
  (|removeTracedMapSigs| untracelist)))
```

———-

51.1.28 defun transTraceItem

[get p**??**]
[member p1042]
[objMode p**??**]
[objVal p**??**]
[domainToGenvar p851]
[unabbrev p**??**]
[constructor? p**??**]
[vecp p**??**]
[transTraceItem p853]
[devaluate p**??**]
[throwKeyedMsg p**??**]
[$doNotAddEmptyModeIfTrue p**??**]

— defun transTraceItem —

```
(defun |transTraceItem| (x)
 (prog (|$doNotAddEmptyModeIfTrue| |value| y)
  (declare (special |$doNotAddEmptyModeIfTrue|))
  (return
   (progn
    (setq |$doNotAddEmptyModeIfTrue| t)
    (cond
     ((atom x)
       (cond
        ((and (setq |value| (|get| x '|value| |$InteractiveFrame|))
              (|member| (|objMode| |value|)
                '((|Mode|) (|Domain|) (|SubDomain| (|Domain|)))))
```

```
          (setq x (|objVal| |value|))
          (cond
           ((setq y (|domainToGenvar| x)) y)
           (t x)))
        ((upper-case-p (elt (princ-to-string x) 0))
          (setq y (|unabbrev| x))
          (cond
           ((|constructor?| y) y)
           ((and (consp y) (|constructor?| (car y))) (car y))
           ((setq y (|domainToGenvar| x)) y)
           (t x)))
        (t x)))
     ((vecp (car x)) (|transTraceItem| (|devaluate| (car x))))
     ((setq y (|domainToGenvar| x)) y)
     (t (|throwKeyedMsg| 's2it0018 (cons x nil))))))))
```

———-

51.1.29 defun removeTracedMapSigs

[$tracedMapSignatures p836]

— **defun removeTracedMapSigs** —

```
(defun |removeTracedMapSigs| (untraceList)
 (declare (special |$tracedMapSignatures|))
 (dolist (name untraceList)
   (remprop name |$tracedMapSignatures|)))
```

———-

51.1.30 defun coerceTraceArgs2E

[spadsysnamep p**??**]
[pname p1039]
[coerceSpadArgs2E p855]
[objValUnwrap p**??**]
[coerceInteractive p**??**]
[objNewWrap p**??**]
[$OutputForm p**??**]
[$mathTraceList p**??**]
[$tracedMapSignatures p836]

— **defun coerceTraceArgs2E** —

```
(defun |coerceTraceArgs2E| (tracename subname args)
 (declare (ignore tracename))
 (let (name)
  (declare (special |$OutputForm| |$mathTraceList| |$tracedMapSignatures|))
  (cond
   ((member (setq name subname) |$mathTraceList|)
     (if (spadsysnamep (pname name))
      (|coerceSpadArgs2E| (reverse (cdr (reverse args))))
      (do ((t1 '(|arg1| |arg2| |arg3| |arg4| |arg5| |arg6| |arg7| |arg8|
                 |arg9| |arg10| |arg11| |arg12| |arg13| |arg14| |arg15|
                 |arg16| |arg17| |arg18| |arg19|) (cdr t1))
           (name nil)
           (t2 args (cdr t2))
           (arg nil)
           (t3 (cdr (lassoc subname |$tracedMapSignatures|)) (cdr t3))
           (type nil)
           (t0 nil))
          ((or (atom t1)
               (progn (setq name (car t1)) nil)
               (atom t2)
               (progn (setq arg (car t2)) nil)
               (atom t3)
               (progn (setq type (car t3)) nil))
             (nreverse0 t0))
          (setq t0
           (cons
            (list '= name
             (|objValUnwrap|
              (|coerceInteractive|
               (|objNewWrap| arg type) |$OutputForm|))) t0)))))
   ((spadsysnamep (pname name)) (reverse (cdr (reverse args))))
   (t args))))
```

———-

51.1.31 defun coerceSpadArgs2E

[seq p??]
[exit p??]
[objValUnwrap p??]
[coerceInteractive p??]
[objNewWrap p??]
[$streamCount p792]
[$OutputForm p??]
[$tracedSpadModemap p??]

— **defun coerceSpadArgs2E** —

```
(defun |coerceSpadArgs2E| (args)
 (let ((|$streamCount| 0))
  (declare (special |$streamCount| |$OutputForm| |$tracedSpadModemap|))
  (do ((t1 '(|arg1| |arg2| |arg3| |arg4| |arg5| |arg6| |arg7| |arg8|
             |arg9| |arg10| |arg11| |arg12| |arg13| |arg14| |arg15|
             |arg16| |arg17| |arg18| |arg19|) (cdr t1))
       (name nil)
       (t2 args (cdr t2))
       (arg nil)
       (t3 (cdr |$tracedSpadModemap|) (cdr t3))
       (type nil)
       (t0 nil))
      ((or (atom t1)
           (progn (setq name (car t1)) nil)
           (atom t2)
           (progn (setq arg (car t2)) nil)
           (atom t3)
           (progn (setq type (car t3)) nil))
        (nreverse0 t0))
    (seq
     (exit
      (setq t0
       (cons
        (cons '=
         (cons name
          (cons (|objValUnwrap|
            (|coerceInteractive|
             (|objNewWrap| arg type)
             |$OutputForm|)) nil)))
        t0)))))))
```

———-

51.1.32 defun subTypes

[lassoc p**??**]
[seq p**??**]
[exit p**??**]
[subTypes p856]

— **defun subTypes** —

```
(defun |subTypes| (|mm| |sublist|)
 (prog (s)
  (return
   (seq
    (cond
```

```
     ((atom |mm|)
       (cond ((setq s (lassoc |mm| |sublist|)) s) (t |mm|)))
     (t
      (prog (t0)
       (setq t0 nil)
       (return
        (do ((t1 |mm| (cdr t1)) (|m| nil))
            ((or (atom t1) (progn (setq |m| (car t1)) nil)) (nreverse0 t0))
         (seq
          (exit
           (setq t0 (cons (|subTypes| |m| |sublist|) t0)))))))))))
```

———-

51.1.33 defun coerceTraceFunValue2E

[spadsysnamep p??]
[pname p1039]
[coerceSpadFunValue2E p858]
[lassoc p??]
[objValUnwrap p??]
[coerceInteractive p??]
[objNewWrap p??]
[$tracedMapSignatures p836]
[$OutputForm p??]
[$mathTraceList p??]

— defun coerceTraceFunValue2E —

```
(defun |coerceTraceFunValue2E| (tracename subname |value|)
 (let (name u)
 (declare (special |$tracedMapSignatures| |$OutputForm| |$mathTraceList|))
   (if (member (setq name subname) |$mathTraceList|)
    (cond
     ((spadsysnamep (pname tracename)) (|coerceSpadFunValue2E| |value|))
     ((setq u (lassoc subname |$tracedMapSignatures|))
       (|objValUnwrap|
        (|coerceInteractive| (|objNewWrap| |value| (car u)) |$OutputForm|)))
     (t |value|))
    |value|)))
```

———-

51.1.34 defun coerceSpadFunValue2E

[objValUnwrap p??]
[coerceInteractive p??]
[objNewWrap p??]
[$streamCount p792]
[$tracedSpadModemap p??]
[$OutputForm p??]

— **defun coerceSpadFunValue2E** —

```
(defun |coerceSpadFunValue2E| (|value|)
 (let (|$streamCount|)
  (declare (special |$streamCount| |$tracedSpadModemap| |$OutputForm|))
  (setq |$streamCount| 0)
  (|objValUnwrap|
   (|coerceInteractive|
    (|objNewWrap| |value| (car |$tracedSpadModemap|))
    |$OutputForm|))))
```

———-

51.1.35 defun isListOfIdentifiers

[seq p??]
[exit p??]
[identp p1040]

— **defun isListOfIdentifiers** —

```
(defun |isListOfIdentifiers| (arg)
 (prog ()
  (return
   (seq
    (prog (t0)
     (setq t0 t)
     (return
      (do ((t1 nil (null t0)) (t2 arg (cdr t2)) (x nil))
          ((or t1 (atom t2) (progn (setq x (car t2)) nil)) t0)
       (seq
        (exit
         (setq t0 (and t0 (identp x))))))))))))
```

———-

51.1.36 defun isListOfIdentifiersOrStrings

[seq p??]
[exit p??]
[identp p1040]

— **defun isListOfIdentifiersOrStrings** —

```
(defun |isListOfIdentifiersOrStrings| (arg)
 (prog ()
  (return
   (seq
    (prog (t0)
     (setq t0 t)
     (return
      (do ((t1 nil (null t0)) (t2 arg (cdr t2)) (x nil))
          ((or t1 (atom t2) (progn (setq x (car t2)) nil)) t0)
       (seq
        (exit
         (setq t0 (and t0 (or (identp x) (stringp x))))))))))))
```

———-

51.1.37 defun getMapSubNames

[get p??]
[union p??]
[getPreviousMapSubNames p860]
[unionq p??]
[$lastUntraced p??]
[$InteractiveFrame p??]
[/tracenames p??]

— **defun getMapSubNames** —

```
(defun |getMapSubNames| (arg)
 (let (lmm subs)
 (declare (special /tracenames |$lastUntraced| |$InteractiveFrame|))
  (setq subs nil)
  (dolist (mapname arg)
   (when (setq lmm (|get| mapname '|localModemap| |$InteractiveFrame|))
     (setq subs
      (append
        (do ((t2 lmm (cdr t2)) (t1 nil) (|mm| nil))
            ((or (atom t2)
                 (progn (setq |mm| (CAR t2)) nil)) (nreverse0 t1))
```

```
          (setq t1 (cons (cons mapname (cadr |mm|)) t1)))
       subs))))
  (|union| subs
    (|getPreviousMapSubNames| (unionq /tracenames |$lastUntraced|)))))
```

———-

51.1.38 defun getPreviousMapSubNames

[get p??]
[exit p??]
[seq p??]

— **defun getPreviousMapSubNames** —

```
(defun |getPreviousMapSubNames| (|traceNames|)
 (prog (lmm subs)
  (return
   (seq
    (progn
     (setq subs nil)
     (seq
      (do ((t0 (assocleft (caar |$InteractiveFrame|)) (cdr t0))
           (mapname nil))
          ((or (atom t0) (progn (setq mapname (car t0)) nil)) nil)
       (seq
        (exit
         (cond
          ((setq lmm
             (|get| mapname '|localModemap| |$InteractiveFrame|))
           (exit
            (cond
             ((member (cadar lmm) |traceNames|)
              (exit
               (do ((t1 lmm (cdr t1)) (|mm| nil))
                   ((or (atom t1) (progn (setq |mm| (car t1)) nil)) nil)
                (seq
                 (exit
                  (setq subs
                   (cons (cons mapname (cadr |mm|)) subs)))))))))))))
      (exit subs)))))))
```

———-

51.1.39 defun lassocSub

[lassq p??]

— **defun lassocSub** —

```
(defun |lassocSub| (x subs)
 (let (y)
  (if (setq y (lassq x subs))
    y
    x)))
```

———-

51.1.40 defun rassocSub

[rassoc p??]

— **defun rassocSub** —

```
(defun |rassocSub| (x subs)
 (let (y)
   (if (setq y (|rassoc| x subs))
    y
    x)))
```

———-

51.1.41 defun isUncompiledMap

[get p??]
[$InteractiveFrame p??]

— **defun isUncompiledMap** —

```
(defun |isUncompiledMap| (x)
 (let (y)
 (declare (special |$InteractiveFrame|))
  (when (setq y (|get| x '|value| |$InteractiveFrame|))
    (and
      (eq (caar y) 'map)
      (null (|get| x '|localModemap| |$InteractiveFrame|))))))
```

———-

51.1.42 defun isInterpOnlyMap

[get p??]
[$InteractiveFrame p??]

— **defun isInterpOnlyMap** —

```
(defun |isInterpOnlyMap| (map)
 (let (x)
 (declare (special |$InteractiveFrame|))
  (when (setq x (|get| map '|localModemap| |$InteractiveFrame|))
    (eq (caaar x) '|interpOnly|))))
```

———-

51.1.43 defun augmentTraceNames

[get p??]
[$InteractiveFrame p??]

— **defun augmentTraceNames** —

```
(defun |augmentTraceNames| (arg)
 (let (mml res)
 (declare (special |$InteractiveFrame|))
 (dolist (tracename arg)
  (if (setq mml (|get| tracename '|localModemap| |$InteractiveFrame|))
   (setq res
    (append
     (prog (t1)
      (setq t1 nil)
      (return
       (do ((t2 mml (cdr t2)) (|mm| nil))
           ((or (atom t2)
                (progn (setq |mm| (CAR t2)) nil))
             (nreverse0 t1))
          (setq t1 (cons (cadr |mm|) t1)))))
    res))
   (setq res (cons tracename res))))
 res))
```

———-

51.1.44 defun isSubForRedundantMapName

[rassocSub p861]
[member p1042]
[assocleft p??]
[$mapSubNameAlist p??]

— **defun isSubForRedundantMapName** —

```
(defun |isSubForRedundantMapName| (subname)
 (let (mapname tail)
 (declare (special |$mapSubNameAlist|))
  (when (setq mapname (|rassocSub| subname |$mapSubNameAlist|))
    (when (setq tail (|member| (cons mapname subname) |$mapSubNameAlist|))
      (member mapname (cdr (assocleft tail)))))))
```

———-

51.1.45 defun untraceMapSubNames

[assocright p??]
[/untrace,2 p??]
[setdifference p??]
[getPreviousMapSubNames p860]
[$mapSubNameAlist p??]
[$lastUntraced p??]

— **defun untraceMapSubNames** —

```
(defun |untraceMapSubNames| (|traceNames|)
 (let (|$mapSubNameAlist| subs)
 (declare (special |$mapSubNameAlist| |$lastUntraced|))
  (if
   (null (setq |$mapSubNameAlist| (|getPreviousMapSubNames| |traceNames|)))
   nil
   (dolist (name (setq subs (assocright |$mapSubNameAlist|)))
     (when (member name /tracenames)
       (|/UNTRACE,2| name nil)
       (setq |$lastUntraced| (setdifference |$lastUntraced| subs)))))))
```

———-

51.1.46 defun funfind,LAM

[qcar p??]
[SEQ p??]
[isFunctor p??]
[exit p??]

— **defun funfind,LAM** —

```
(defun |funfind,LAM| (functor opname)
 (prog (ops tmp1)
  (return
   (seq
    (progn
     (setq ops (|isFunctor| functor))
     (prog (t0)
      (setq t0 nil)
      (return
       (do ((t1 ops (cdr t1)) (u nil))
           ((or (atom t1) (progn (setq u (car t1)) nil)) (nreverse0 t0))
        (seq
         (exit
          (cond
           ((and (consp u)
                 (progn
                  (setq tmp1 (qcar u))
                  (and (consp tmp1) (equal (qcar tmp1) opname))))
             (setq t0 (cons u t0)))))))))))))
```

———-

51.1.47 defmacro funfind

— **defmacro funfind** —

```
(defmacro |funfind| (&whole t0 &rest notused &aux t1)
 (declare (ignore notused))
 (dsetq t1 t0)
 (cons '|funfind,LAM| (wrap (cdr t1) '(quote quote))))
```

———-

51.1.48 defun isDomainOrPackage

[refvecp p??]
[poundsign p??]
[isFunctor p??]
[opOf p??]

— **defun isDomainOrPackage** —

```
(defun |isDomainOrPackage| (dom)
 (and
  (refvecp dom)
  (> (|#| dom) 0)
  (|isFunctor| (|opOf| (elt dom 0)))))
```

———-

51.1.49 defun isTraceGensym

[gensymp p??]

— **defun isTraceGensym** —

```
(defun |isTraceGensym| (x)
 (gensymp x))
```

———-

51.1.50 defun spadTrace,g

— **defun spadTrace,g** —

```
(defun |spadTrace,g| (x)
  (if (stringp x) (intern x) x))
```

———-

51.1.51 defun spadTrace,isTraceable

[seq p??]
[exit p??]

[gensymp p??]
[reportSpadTrace p882]
[bpiname p??]

— **defun spadTrace,isTraceable** —

```
(defun |spadTrace,isTraceable| (x |domain|)
 (prog (n |functionSlot|)
  (return
   (seq
    (progn
     (setq n (caddr x))
     x
     (seq
      (if (atom (elt |domain| n)) (exit nil))
      (setq |functionSlot| (car (elt |domain| n)))
      (if (gensymp |functionSlot|)
       (exit (seq (|reportSpadTrace| '|Already Traced| x) (exit nil))))
      (if (null (bpiname |functionSlot|))
       (exit
        (seq
         (|reportSpadTrace| '|No function for| x)
         (exit nil))))
      (exit t)))))))
```

———-

51.1.52 defun spadTrace

[refvecp p??]
[aldorTrace p??]
[isDomainOrPackage p865]
[userError p??]
[seq p??]
[exit p??]
[spadTrace,g p865]
[getOption p882]
[removeOption p851]
[opOf p??]
[assoc p??]
[kdr p??]
[flattenOperationAlist p873]
[getOperationAlistFromLisplib p??]
[spadTrace,isTraceable p865]
[as-insert p??]
[bpiname p??]

[spadTraceAlias p881]
[subTypes p856]
[constructSubst p432]
[bpitrace p**??**]
[rplac p**??**]
[printDashedLine p**??**]
[reportSpadTrace p882]
[setletprintflag p**??**]
[spadReply p885]
[$tracedModemap p**??**]
[$fromSpadTrace p**??**]
[$letAssoc p**??**]
[$reportSpadTrace p882]
[$traceNoisely p836]
[/tracenames p**??**]

— defun spadTrace —

```
(defun |spadTrace| (domain options)
 (let (|$tracedModemap| listofoperations listofvariables
       listofbreakvars anyiftrue domainid currententry
       currentalist opstructurelist sig kind triple fn op
       mm n alias tracename sigslotnumberalist)
 (declare (special |$tracedModemap| /tracenames |$fromSpadTrace| |$letAssoc|
    |$reportSpadTrace| |$traceNoisely|))
 (setq |$fromSpadTrace| t)
 (setq |$tracedModemap| nil)
 (cond
  ((and (consp domain)
        (refvecp (car domain))
        (eql (elt (car domain) 0) 0))
    (|aldorTrace| domain options))
  ((null (|isDomainOrPackage| domain))
    (|userError| "bad argument to trace"))
  (t
   (setq listofoperations
    (prog (t0)
     (setq t0 nil)
     (return
      (do ((t1 (|getOption| 'ops options) (cdr t1)) (x nil))
          ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
       (seq
        (exit
         (setq t0 (cons (|spadTrace,g| x) t0))))))))
   (cond
    ((setq listofvariables (|getOption| 'vars options))
      (setq options (|removeOption| 'vars options))))
   (cond
```

```
 ((setq listofbreakvars (|getOption| 'varbreak options))
   (setq options (|removeOption| 'varbreak options))))
(setq anyiftrue (null listofoperations))
(setq domainid (|opOf| (elt domain 0)))
(setq currententry (|assoc| domain /tracenames))
(setq currentalist (kdr currententry))
(setq opstructurelist
 (|flattenOperationAlist| (|getOperationAlistFromLisplib| domainid)))
(setq sigslotnumberalist
 (prog (t2)
  (setq t2 nil)
  (return
   (do ((t3 opstructurelist (cdr t3)) (t4 nil))
       ((or (atom t3)
            (progn (setq t4 (CAR t3)) nil)
            (progn
             (progn
              (setq op (car t4))
              (setq sig (cadr t4))
              (setq n (caddr t4))
              (setq kind (car (cddddr t4))) t4)
             nil))
        (nreverse0 t2))
    (seq
     (exit
      (cond
       ((and (eq kind 'elt)
             (or anyiftrue (member op listofoperations))
             (integerp n)
             (|spadTrace,isTraceable|
              (setq triple
               (cons op (cons sig (cons n nil)))) domain))
        (setq t2 (cons triple t2)))))))))
(cond
 (listofvariables
  (do ((t5 sigslotnumberalist (cdr t5)) (t6 nil))
      ((or (atom t5)
           (progn (setq t6 (car t5)) nil)
           (progn (progn (setq n (caddr t6)) t6) nil))
       nil)
   (seq
    (exit
     (progn
      (setq fn (car (elt domain n)))
      (setq |$letAssoc|
       (as-insert (bpiname fn) listofvariables |$letAssoc|)))))))))
(cond
 (listofbreakvars
  (do ((t7 sigslotnumberalist (cdr t7)) (t8 nil))
      ((or (atom t7)
```

```
            (progn (setq t8 (car t7)) nil)
            (progn (progn (setq n (caddr t8)) t8) nil))
           nil)
     (seq
      (exit
       (progn
        (setq fn (car (elt domain n)))
        (setq |$letAssoc|
         (as-insert (bpiname fn)
          (cons (cons 'break listofbreakvars) nil) |$letAssoc|))))))))
  (do ((t9 sigslotnumberalist (cdr t9)) (|pair| nil))
      ((or (atom t9)
           (progn (setq |pair| (car t9)) nil)
           (progn
            (progn
             (setq op (car |pair|))
             (setq mm (cadr |pair|))
             (setq n (caddr |pair|))
             |pair|)
            nil))
          nil)
   (seq
    (exit
     (progn
      (setq alias (|spadTraceAlias| domainid op n))
      (setq |$tracedModemap|
        (|subTypes| mm (|constructSubst| (elt domain 0))))
      (setq tracename
        (bpitrace (car (elt domain n)) alias options))
      (nconc |pair|
        (cons listofvariables
         (cons (car (elt domain n))
          (cons tracename (cons alias nil)))))
      (rplac (car (elt domain n)) tracename)))))
  (setq sigslotnumberalist
   (prog (t10)
    (setq t10 nil)
    (return
     (do ((t11 sigslotnumberalist (cdr t11)) (x nil))
         ((or (atom t11) (progn (setq x (car t11)) nil)) (nreverse0 t10))
      (seq
       (exit
        (cond ((cdddr x) (setq t10 (cons x t10)))))))))))
  (cond
   (|$reportSpadTrace|
    (cond (|$traceNoisely| (|printDashedLine|)))
    (do ((t12 (|orderBySlotNumber| sigslotnumberalist) (cdr t12))
         (x nil))
        ((or (atom t12)
             (progn (setq x (car t12)) nil))
```

```
            nil)
          (seq (exit (|reportSpadTrace| 'tracing x))))))
   (cond (|$letAssoc| (setletprintflag t)))
   (cond
    (currententry
     (rplac (cdr currententry)
       (append sigslotnumberalist currentalist)))
    (t
     (setq /tracenames
      (cons (cons domain sigslotnumberalist) /tracenames))
       (|spadReply|)))))))
```

———-

51.1.53 defun traceDomainLocalOps

[sayMSG p329]

— **defun traceDomainLocalOps** —

```
(defun |traceDomainLocalOps| ()
  (|sayMSG| '("  The )local option has been withdrawn"))
  (|sayMSG| '("  Use )ltr to trace local functions.")))
```

———-

51.1.54 defun untraceDomainLocalOps

[sayMSG p329]

— **defun untraceDomainLocalOps** —

```
(defun |untraceDomainLocalOps| ()
  (|sayMSG| '("  The )local option has been withdrawn"))
  (|sayMSG| '("  Use )ltr to trace local functions.")))
```

———-

51.1.55 defun traceDomainConstructor

[getOption p882]
[seq p**??**]
[exit p**??**]

[spadTrace p866]
[concat p1041]
[embed p??]
[mkq p??]
[loadFunctor p1032]
[traceDomainLocalOps p870]
[$ConstructorCache p??]

— defun traceDomainConstructor —

```
(defun |traceDomainConstructor| (domainConstructor options)
 (prog (listOfLocalOps argl domain innerDomainConstructor)
 (declare (special |$ConstructorCache|))
  (return
   (seq
    (progn
     (|loadFunctor| domainConstructor)
     (setq listOfLocalOps (|getOption| 'local options))
     (when listOfLocalOps (|traceDomainLocalOps|))
     (cond
      ((and listOfLocalOps (null (|getOption| 'ops options))) nil)
      (t
       (do ((t2 (hget |$ConstructorCache| domainConstructor) (cdr t2))
            (t3 nil))
           ((or (atom t2)
                (progn (setq t3 (car t2)) nil)
                (progn
                 (progn
                  (setq argl (car t3))
                  (setq domain (cddr t3)) t3)
                 nil))
               nil)
         (seq
          (exit
           (|spadTrace| domain options))))
       (setq /tracenames (cons domainConstructor /tracenames))
       (setq innerDomainConstructor
         (intern (concat domainConstructor ";")))
       (cond
        ((fboundp innerDomainConstructor)
          (setq domainConstructor innerDomainConstructor)))
       (embed domainConstructor
        (cons 'lambda
         (cons
          (cons '&rest
           (cons 'args nil))
          (cons
           (cons 'prog
            (cons
```

```
            (cons 'domain nil)
            (cons
             (cons 'setq
              (cons 'domain
               (cons
                (cons 'apply (cons domainConstructor
                 (cons 'args nil))) nil)))
             (cons
              (cons '|spadTrace|
               (cons 'domain
                (cons (mkq options) nil)))
              (cons (cons 'return (cons 'domain nil)) nil)))))
          nil)))))))))))
```

———-

51.1.56 defun untraceDomainConstructor,keepTraced?

[seq p??]
[qcar p??]
[isDomainOrPackage p865]
[boot-equal p??]
[kar p??]
[devaluate p??]
[exit p??]
[/untrace,0 p??]

— **defun untraceDomainConstructor,keepTraced?** —

```
(defun |untraceDomainConstructor,keepTraced?| (df domainConstructor)
 (prog (dc)
  (return
   (seq
    (if (and
         (and
          (and (consp df) (progn (setq dc (qcar df)) t))
           (|isDomainOrPackage| dc))
         (boot-equal (kar (|devaluate| dc)) domainConstructor))
     (exit (seq (|/UNTRACE,0| (cons dc nil)) (exit nil))))
    (exit t)))))
```

———-

51.1.57 defun untraceDomainConstructor

[untraceDomainConstructor,keepTraced? p872]
[unembed p??]
[seq p??]
[exit p??]
[concat p1041]
[delete p??]
[/tracenames p??]

— **defun untraceDomainConstructor** —

```
(defun |untraceDomainConstructor| (domainConstructor)
 (prog (innerDomainConstructor)
 (declare (special /tracenames))
  (return
   (seq
    (progn
     (setq /tracenames
      (prog (t0)
       (setq t0 nil)
       (return
        (do ((t1 /tracenames (cdr t1)) (df nil))
            ((or (atom t1) (progn (setq df (car t1)) nil)) (nreverse0 t0))
         (seq
          (exit
           (cond ((|untraceDomainConstructor,keepTraced?|
                    df domainConstructor)
             (setq t0 (cons df t0)))))))))))
     (setq innerDomainConstructor
      (intern (concat domainConstructor ";")))
     (cond
      ((fboundp innerDomainConstructor) (unembed innerDomainConstructor))
      (t (unembed domainConstructor)))
     (setq /tracenames (|delete| domainConstructor /tracenames)))))))
```

———-

51.1.58 defun flattenOperationAlist

[seq p??]
[exit p??]

— **defun flattenOperationAlist** —

```
(defun |flattenOperationAlist| (|opAlist|)
 (prog (op |mmList| |res|)
```

```
  (return
   (seq
    (progn
     (setq |res| nil)
     (do ((t0 |opAlist| (cdr t0)) (t1 nil))
         ((or (atom t0)
              (progn (setq t1 (car t0)) nil)
              (progn
               (progn (setq op (car t1)) (setq |mmList| (cdr t1)) t1)
               nil))
             nil)
      (seq
       (exit
        (setq |res|
         (append |res|
          (prog (t2)
           (setq t2 nil)
           (return
            (do ((t3 |mmList| (cdr t3)) (mm nil))
                ((or (atom t3)
                     (progn (setq mm (car t3)) nil)) (nreverse0 t2))
             (seq
              (exit
               (setq t2 (cons (cons op mm) t2)))))))))))
     |res|)))))
```

———-

51.1.59 defun mapLetPrint

[getAliasIfTracedMapParameter p879]
[getBpiNameIfTracedMap p880]
[letPrint p875]

— **defun mapLetPrint** —

```
(defun |mapLetPrint| (x val currentFunction)
  (setq x (|getAliasIfTracedMapParameter| x currentFunction))
  (setq currentFunction (|getBpiNameIfTracedMap| currentFunction))
  (|letPrint| x val currentFunction))
```

———-

51.1.60 defun letPrint

[lassoc p??]
[isgenvar p876]
[isSharpVarWithNum p876]
[gensymp p??]
[sayBrightlyNT p??]
[bright p??]
[shortenForPrinting p881]
[hasPair p881]
[pname p1039]
[break p896]
[$letAssoc p??]

— **defun letPrint** —

```
(defun |letPrint| (x |val| |currentFunction|)
 (prog (y)
 (declare (special |$letAssoc|))
  (return
   (progn
    (cond ((and |$letAssoc|
                (or
                 (setq y (lassoc |currentFunction| |$letAssoc|))
                 (setq y (lassoc '|all| |$letAssoc|))))
     (cond
      ((and (or (eq y '|all|)
                 (member x y))
            (null
             (or (isgenvar x) (|isSharpVarWithNum| x) (gensymp x))))
        (|sayBrightlyNT| (append (|bright| x) (cons '|: | nil)))
        (prin1 (|shortenForPrinting| |val|))
        (terpri)))
     (cond
      ((and (setq y (|hasPair| 'break y))
            (or (eq y '|all|)
                (and (member x y)
                     (null (member (elt (pname x) 0) '($ |#|)))
                     (null (gensymp x)))))
       (|break|
        (append
         (|bright| |currentFunction|)
         (cons "breaks after"
          (append
           (|bright| x)
           (cons ":= " (cons (|shortenForPrinting| |val|) nil)))))))
      (t nil))))
    |val|))))
```

———-

51.1.61 defun Identifier beginning with a sharpsign-number?

This tests if x is an identifier beginning with # followed by a number. [isSharpVar p876]
[pname p1039]
[qcsize p??]
[digitp p1039]
[dig2fix p??]

— **defun isSharpVarWithNum** —

```
(defun |isSharpVarWithNum| (x)
 (let (p n d ok c)
  (cond
   ((null (|isSharpVar| x)) nil)
   ((> 2 (setq n (qcsize (setq p (pname x))))) nil)
   (t
    (setq ok t)
    (setq c 0)
    (do ((t1 (1- n)) (i 1 (1+ i)))
        ((or (> i t1) (null ok)) nil)
      (setq d (elt p i))
      (when (setq ok (digitp d))
        (setq c (+ (* 10 c) (dig2fix d)))))
    (when ok c)))))
```

———-

51.1.62 defun Identifier beginning with a sharpsign?

This tests if x is an identifier beginning with # [identp p1040]

— **defun isSharpVar** —

```
(defun |isSharpVar| (x)
 (and (identp x) (char= (schar (symbol-name x) 0) #\#)))
```

———-

51.1.63 defun isgenvar

[size p1039]
[digitp p1039]

[identp p1040]

— **defun isgenvar** —

```
(defun isgenvar (x)
 (and (identp x)
  (let ((y (symbol-name x)))
   (and (char= #\$ (elt y 0)) (> (size y) 1) (digitp (elt y 1))))))
```

———-

51.1.64 defun letPrint2

[letPrint2 p877]
[lassoc p**??**]
[isgenvar p876]
[isSharpVarWithNum p876]
[gensymp p**??**]
[mathprint p**??**]
[print p**??**]
[hasPair p881]
[pname p1039]
[break p896]
[bright p**??**]
[$BreakMode p653]
[$letAssoc p**??**]

— **defun letPrint2** —

```
(defun |letPrint2| (x |printform| |currentFunction|)
 (prog (|$BreakMode| |flag| y)
  (declare (special |$BreakMode| |$letAssoc|))
  (return
   (progn
    (setq |$BreakMode| nil)
    (cond
     ((and |$letAssoc|
           (or (setq y (lassoc |currentFunction| |$letAssoc|))
               (setq y (lassoc '|all| |$letAssoc|))))
      (cond
       ((and
          (or (eq y '|all|) (member x y))
          (null (or (isgenvar x) (|isSharpVarWithNum| x) (gensymp x))))
        (setq |$BreakMode| '|letPrint2|)
        (setq |flag| nil)
        (catch '|letPrint2|
```

```
         (|mathprint| (cons '= (cons x (cons |printform| nil)))) |flag|)
        (cond
         ((eq |flag| '|letPrint2|) (|print| |printform|))
         (t nil))))
     (cond
      ((and
        (setq y (|hasPair| 'break y))
        (or (eq y '|all|)
            (and
             (member x y)
             (null (member (elt (pname x) 0) '($ |#|)))
             (null (gensymp x)))))
       (|break|
        (append
         (|bright| |currentFunction|)
         (cons "breaks after"
          (append (|bright| x) (cons '|:= | (cons |printform| nil)))))))
      (t nil))))
   x))))
```

———-

51.1.65 defun letPrint3

This is the version for use when we have our hands on a function to convert the data into type "Expression" [letPrint2 p877]
[lassoc p**??**]
[isgenvar p876]
[isSharpVarWithNum p876]
[gensymp p**??**]
[mathprint p**??**]
[spadcall p**??**]
[print p**??**]
[hasPair p881]
[pname p1039]
[break p896]
[bright p**??**]
[$BreakMode p653]
[$letAssoc p**??**]

— defun letPrint3 —

```
(defun |letPrint3| (x |xval| |printfn| |currentFunction|)
 (prog (|$BreakMode| |flag| y)
  (declare (special |$BreakMode| |$letAssoc|))
  (return
```

```
(progn
 (setq |$BreakMode| nil)
 (cond
  ((and |$letAssoc|
    (or (setq y (lassoc |currentFunction| |$letAssoc|))
        (setq y (lassoc '|all| |$letAssoc|))))
   (cond
    ((and
       (or (eq y '|all|) (member x y))
       (null (or (isgenvar x) (|isSharpVarWithNum| x) (gensymp x))))
      (setq |$BreakMode| '|letPrint2|)
      (setq |flag| nil)
      (catch '|letPrint2|
       (|mathprint|
        (cons '= (cons x (cons (spadcall |xval| |printfn|) nil))))
       |flag|)
      (cond
       ((eq |flag| '|letPrint2|) (|print| |xval|))
       (t nil))))
   (cond
    ((and
      (setq y (|hasPair| 'break y))
      (or
       (eq y '|all|)
       (and
        (member x y)
        (null (member (elt (pname x) 0) '($ |#|)))
        (null (gensymp x)))))
      (|break|
       (append
        (|bright| |currentFunction|)
        (cons "breaks after"
         (append (|bright| x) (cons ":= " (cons |xval| nil)))))))
    (t nil))))
 x))))
```

———-

51.1.66 defun getAliasIfTracedMapParameter

[isSharpVarWithNum p876]
[get p??]
[exit p??]
[spaddifference p??]
[string2pint-n p??]
[substring p??]
[pname p1039]

[seq p**??**]
[$InteractiveFrame p**??**]

— defun getAliasIfTracedMapParameter —

```
(defun |getAliasIfTracedMapParameter| (x |currentFunction|)
 (prog (|aliasList|)
 (declare (special |$InteractiveFrame|))
  (return
   (seq
    (cond
     ((|isSharpVarWithNum| x)
       (cond
        ((setq |aliasList|
           (|get| |currentFunction| 'alias |$InteractiveFrame|))
         (exit
          (elt |aliasList|
           (spaddifference
            (string2pint-n (substring (pname x) 1 nil) 1) 1))))))
     (t x))))))
```

———-

51.1.67 defun getBpiNameIfTracedMap

[get p**??**]
[exit p**??**]
[seq p**??**]
[$InteractiveFrame p**??**]
[/tracenames p**??**]

— defun getBpiNameIfTracedMap —

```
(defun |getBpiNameIfTracedMap| (name)
 (prog (lmm bpiName)
 (declare (special |$InteractiveFrame| /tracenames))
  (return
   (seq
    (cond
     ((setq lmm (|get| name '|localModemap| |$InteractiveFrame|))
       (cond
        ((member (setq bpiName (cadar lmm)) /tracenames)
           (exit bpiName))))
     (t name))))))
```

———-

51.1.68 defun hasPair

[qcar p??]
[qcdr p??]
[hasPair p881]

— **defun hasPair** —

```
(defun |hasPair| (key arg)
 (prog (tmp1 a)
  (return
   (cond
    ((atom arg) nil)
    ((and (consp arg)
          (progn
           (setq tmp1 (qcar arg))
           (and (consp tmp1)
                (equal (qcar tmp1) key)
                (progn (setq a (qcdr tmp1)) t))))
       a)
    (t (|hasPair| key (cdr arg)))))))
```

———-

51.1.69 defun shortenForPrinting

[isDomainOrPackage p865]
[devaluate p??]

— **defun shortenForPrinting** —

```
(defun |shortenForPrinting| (|val|)
 (if (|isDomainOrPackage| |val|)
  (|devaluate| |val|)
  |val|))
```

———-

51.1.70 defun spadTraceAlias

[internl p??]

— **defun spadTraceAlias** —

```
(defun |spadTraceAlias| (domainid op n)
```

```
 (internl domainid (intern "." "boot") op '|,| (princ-to-string n)))
```

———-

51.1.71 defun getOption

[assoc p**??**]

— **defun getOption** —

```
(defun |getOption| (opt l)
 (let (y)
  (when (setq y (|assoc| opt l))  (cdr y))))
```

———-

51.1.72 defun reportSpadTrace

[qcar p**??**]
[sayBrightly p**??**]
[$traceNoisely p836]

— **defun reportSpadTrace** —

```
(defun |reportSpadTrace| (|header| t0)
 (prog (op sig n |t| |msg| |namePart| y |tracePart|)
 (declare (special |$traceNoisely|))
  (return
   (progn
    (setq op (car t0))
    (setq sig (cadr t0))
    (setq n (caddr t0))
    (setq |t| (cdddr t0))
    (cond
     ((null |$traceNoisely|) nil)
     (t
      (setq |msg|
       (cons |header|
        (cons '|%b|
         (cons op
          (cons '|:|
           (cons '|%d|
            (cons (CDR sig)
             (cons '| -> |
              (cons (car sig)
```

```
            (cons '| in slot |
             (cons n nil))))))))))))
      (setq |namePart| nil)
      (setq |tracePart|
       (cond
        ((and (consp |t|) (progn (setq y (qcar |t|)) t) (null (null y)))
          (cond
           ((eq y '|all|)
             (cons '|%b| (cons '|all| (cons '|%d| (cons '|vars| nil)))))
           (t (cons '| vars: | (cons y nil)))))
        (t nil)))
      (|sayBrightly| (append |msg| (append |namePart| |tracePart|)))))))))
```

———-

51.1.73 defun orderBySlotNumber

[seq p??]
[assocright p??]
[orderList p??]
[exit p??]

— **defun orderBySlotNumber** —

```
(defun |orderBySlotNumber| (arg)
 (prog (n)
  (return
   (seq
    (assocright
     (|orderList|
      (prog (t0)
       (setq t0 nil)
       (return
        (do ((t1 arg (cdr t1)) (x nil))
            ((or (atom t1)
                 (progn (setq x (car t1)) nil)
                 (progn (progn (setq n (caddr x)) x) nil))
             (nreverse0 t0))
         (seq
          (exit
           (setq t0 (cons (cons n x) t0)))))))))))))
```

———-

51.1.74 defun /tracereply

[qcar p**??**]
[isDomainOrPackage p865]
[devaluate p**??**]
[seq p**??**]
[exit p**??**]
[/tracenames p**??**]

— **defun /tracereply** —

```
(defun /tracereply ()
 (prog (|d| domainlist |functionList|)
 (declare (special /tracenames))
  (return
   (seq
    (cond
     ((null /tracenames) "   Nothing is traced.")
     (t
       (do ((t0 /tracenames (cdr t0)) (x nil))
           ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
        (seq
         (exit
          (cond
           ((and (consp x)
                 (progn (setq |d| (qcar x)) t)
                 (|isDomainOrPackage| |d|))
              (setq domainlist (cons (|devaluate| |d|) domainlist)))
           (t
            (setq |functionList| (cons x |functionList|))))))))
       (append |functionList|
        (append domainlist (cons '|traced| nil)))))))))
```

———-

51.1.75 defun spadReply,printName

[seq p**??**]
[qcar p**??**]
[isDomainOrPackage p865]
[exit p**??**]
[devaluate p**??**]

— **defun spadReply,printName** —

```
(defun |spadReply,printName| (x)
```

```
(prog (|d|)
 (return
  (seq
   (if (and (and (consp x) (progn (setq |d| (qcar x)) t))
            (|isDomainOrPackage| |d|))
     (exit (|devaluate| |d|)))
    (exit x)))))
```

———-

51.1.76 defun spadReply

[seq p??]
[exit p??]
[spadReply,printName p884]
[/tracenames p??]

— **defun spadReply** —

```
(defun |spadReply| ()
 (prog ()
 (declare (special /tracenames))
  (return
   (seq
    (prog (t0)
     (setq t0 nil)
     (return
       (do ((t1 /tracenames (cdr t1)) (x nil))
           ((or (atom t1) (progn (setq x (car t1)) nil)) (nreverse0 t0))
        (seq
         (exit
          (setq t0 (cons (|spadReply,printName| x) t0))))))))))
```

———-

51.1.77 defun spadUntrace

[isDomainOrPackage p865]
[userError p??]
[getOption p882]
[devaluate p??]
[assoc p??]
[sayMSG p329]
[bright p??]

[prefix2String p??]
[bpiname p??]
[remover p887]
[setletprintflag p??]
[bpiuntrace p??]
[rplac p??]
[seq p??]
[exit p??]
[delasc p??]
[spadReply p885]
[$letAssoc p??]
[/tracenames p??]

— defun spadUntrace —

```
(defun |spadUntrace| (domain options)
 (prog (anyiftrue listofoperations domainid |pair| sigslotnumberalist
        op sig n |lv| |bpiPointer| tracename alias |assocPair|
        |newSigSlotNumberAlist|)
 (declare (special |$letAssoc| /tracenames))
  (return
   (seq
    (cond
     ((null (|isDomainOrPackage| domain))
       (|userError| "bad argument to untrace"))
     (t
      (setq anyiftrue (null options))
      (setq listofoperations (|getOption| '|ops:| options))
      (setq domainid (|devaluate| domain))
      (cond
       ((null (setq |pair| (|assoc| domain /tracenames)))
         (|sayMSG|
          (cons "   No functions in"
           (append
            (|bright| (|prefix2String| domainid))
            (cons "are now traced." nil)))))
       (t
        (setq sigslotnumberalist (cdr |pair|))
        (do ((t0 sigslotnumberalist (cdr t0)) (|pair| nil))
            ((or (atom t0)
                 (progn (setq |pair| (car t0)) nil)
                 (progn
                  (progn
                   (setq op (car |pair|))
                   (setq sig (cadr |pair|))
                   (setq n (caddr |pair|))
                   (setq |lv| (cadddr |pair|))
                   (setq |bpiPointer| (car (cddddr |pair|)))
                   (setq tracename (cadr (cddddr |pair|)))
```

```
                (setq alias (caddr (cddddr |pair|)))
                |pair|)
               nil))
              nil)
         (seq
          (exit
           (cond
            ((or anyiftrue (member op listofoperations))
              (progn
                (bpiuntrace tracename alias)
                (rplac (car (elt domain n)) |bpiPointer|)
                (rplac (cdddr |pair|) nil)
                (cond
                 ((setq |assocPair|
                     (|assoc| (bpiname |bpiPointer|) |$letAssoc|))
                   (setq |$letAssoc| (remover |$letAssoc| |assocPair|))
                   (cond
                    ((null |$letAssoc|) (setletprintflag nil))
                    (t nil)))
                 (t nil))))))))
        (setq |newSigSlotNumberAlist|
         (prog (t1)
          (setq t1 nil)
          (return
           (do ((t2 sigslotnumberalist (cdr t2)) (x nil))
               ((or (atom t2) (progn (setq x (car t2)) nil)) (nreverse0 t1))
             (seq
              (exit
               (cond ((cdddr x) (setq t1 (cons x t1)))))))))))
        (cond
         (|newSigSlotNumberAlist|
          (rplac (cdr |pair|) |newSigSlotNumberAlist|))
         (t
          (setq /tracenames (delasc domain /tracenames))
          (|spadReply|))))))))))))
```

———-

51.1.78 defun remover

[remover p887]

— **defun remover** —

```
(defun remover (lst item)
 (cond
  ((null (consp lst)) (cond ((equal lst item) nil) (t lst)))
  ((equal (car lst) item) (cdr lst))
```

```
  (t
   (rplnode lst (remover (car lst) item) (remover (cdr lst) item))
   (rplaca lst (remover (car lst) item))
   (rplacd lst (remover (cdr lst) item))
   lst)))
```

———-

51.1.79 defun prTraceNames,fn

[seq p??]
[qcar p??]
[qcdr p??]
[isDomainOrPackage p865]
[exit p??]
[devaluate p??]

— **defun prTraceNames,fn** —

```
(defun |prTraceNames,fn| (x)
 (prog (|d| |t|)
  (return
   (seq
    (if (and (and (consp x)
                  (progn (setq |d| (qcar x)) (setq |t| (qcdr x)) t))
             (|isDomainOrPackage| |d|))
      (exit (cons (|devaluate| |d|) |t|)))
    (exit x)))))
```

———-

51.1.80 defun prTraceNames

[seq p??]
[exit p??]
[prTraceNames,fn p888]
[/tracenames p??]

— **defun prTraceNames** —

```
(defun |prTraceNames| ()
 (declare (special /tracenames))
 (seq
  (progn
```

```
   (do ((t0 /tracenames (cdr t0)) (x nil))
       ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
    (seq
     (exit
      (print (|prTraceNames,fn| x))))) nil)))
```

———-

51.1.81 defvar $constructors

— **initvars** —

```
(defvar |$constructors| nil)
```

———-

51.1.82 defun traceReply

[sayMessage p**??**]
[sayBrightly p**??**]
[qcar p**??**]
[isDomainOrPackage p865]
[addTraceItem p892]
[isFunctor p**??**]
[isgenvar p876]
[userError p**??**]
[seq p**??**]
[exit p**??**]
[isSubForRedundantMapName p863]
[rassocSub p861]
[poundsign p**??**]
[sayMSG p329]
[sayBrightlyLength p**??**]
[flowSegmentedMsg p**??**]
[concat p1041]
[prefix2String p**??**]
[abbreviate p**??**]
[$domains p**??**]
[$packages p**??**]
[$constructors p889]
[$linelength p765]
[/tracenames p**??**]

— defun traceReply —

```
(defun |traceReply| ()
 (prog (|$domains| |$packages| |$constructors| |d| |functionList|
        |displayList|)
  (declare (special |$domains| |$packages| |$constructors| /tracenames
     $linelength))
  (return
   (seq
    (progn
     (setq |$domains| nil)
     (setq |$packages| nil)
     (setq |$constructors| nil)
     (cond
      ((null /tracenames) (|sayMessage| "   Nothing is traced now."))
      (t
        (|sayBrightly| " ")
        (do ((t0 /tracenames (cdr t0)) (x nil))
            ((or (atom t0) (progn (setq x (car t0)) nil)) nil)
         (seq
          (exit
           (cond
            ((and (consp x)
                  (progn (setq |d| (qcar x)) t) (|isDomainOrPackage| |d|))
              (|addTraceItem| |d|))
            ((atom x)
               (cond
                ((|isFunctor| x) (|addTraceItem| x))
                ((isgenvar x) (|addTraceItem| (EVAL x)))
                (t (setq |functionList| (cons x |functionList|)))))
            (t (|userError| "bad argument to trace"))))))
        (setq |functionList|
          (prog (t1)
           (setq t1 nil)
           (return
            (do ((t2 |functionList| (cdr t2)) (x nil))
                ((or (atom t2) (progn (setq x (car t2)) nil)) t1)
              (seq
               (exit
                (cond
                 ((null (|isSubForRedundantMapName| x))
                   (setq t1
                    (append t1
                     (cons (|rassocSub| x |$mapSubNameAlist|)
                           (cons " " nil))))))))))))
       (cond
        (|functionList|
         (cond
          ((eql 2 (|#| |functionList|))
```

```
      (|sayMSG| (cons '|   Function traced: | |functionList|)))
   ((<= (+ 22 (|sayBrightlyLength| |functionList|)) $linelength)
      (|sayMSG| (cons '|   Functions traced: | |functionList|)))
   (t
      (|sayBrightly| "   Functions traced:")
      (|sayBrightly|
       (|flowSegmentedMsg| |functionList| $linelength 6))))))
(cond
 (|$domains|
  (setq |displayList|
   (|concat|
    (|prefix2String| (CAR |$domains|))
    (prog (t3)
     (setq t3 nil)
     (return
      (do ((t4 (cdr |$domains|) (cdr t4)) (x nil))
          ((or (atom t4) (progn (setq x (car t4)) nil)) t3)
       (seq
        (exit
         (setq t3
          (append t3 (|concat| "," " " (|prefix2String| x))))))))))))
  (cond
   ((atom |displayList|)
    (setq |displayList| (cons |displayList| nil))))
  (|sayBrightly| "   Domains traced: ")
  (|sayBrightly| (|flowSegmentedMsg| |displayList| $linelength 6))))
(cond
 (|$packages|
   (setq |displayList|
    (|concat|
     (|prefix2String| (CAR |$packages|))
     (prog (t5)
      (setq t5 nil)
      (return
       (do ((t6 (cdr |$packages|) (cdr t6)) (x nil))
           ((or (atom t6) (progn (setq x (car t6)) nil)) t5)
        (seq
         (exit
          (setq t5
           (append t5 (|concat| '|, | (|prefix2String| x))))))))))))
   (cond ((atom |displayList|)
       (setq |displayList| (cons |displayList| nil))))
   (|sayBrightly| "   Packages traced: ")
   (|sayBrightly| (|flowSegmentedMsg| |displayList| $linelength 6))))
(cond
 (|$constructors|
  (setq |displayList|
   (|concat|
    (|abbreviate| (CAR |$constructors|))
    (prog (t7)
```

```
          (setq t7 nil)
           (return
            (do ((t8 (cdr |$constructors|) (cdr t8)) (x nil))
                ((or (atom t8) (progn (setq x (car t8)) nil)) t7)
             (seq
              (exit
               (setq t7
               (append t7 (|concat| '|, | (|abbreviate| x))))))))))
       (cond ((atom |displayList|)
          (setq |displayList| (cons |displayList| nil))))
       (|sayBrightly| "   Parameterized constructors traced:")
       (|sayBrightly| (|flowSegmentedMsg| |displayList| $linelength 6)))
      (t nil)))))))))
```

———-

51.1.83 defun addTraceItem

[constructor? p**??**]
[isDomain p**??**]
[devaluate p**??**]
[isDomainOrPackage p865]
[$constructors p889]
[$domains p**??**]
[$packages p**??**]

— **defun addTraceItem** —

```
(defun |addTraceItem| (|d|)
 (declare (special |$constructors| |$domains| |$packages|))
 (cond
  ((|constructor?| |d|)
    (setq |$constructors| (cons |d| |$constructors|)))
  ((|isDomain| |d|)
    (setq |$domains| (cons (|devaluate| |d|) |$domains|)))
  ((|isDomainOrPackage| |d|)
    (setq |$packages| (cons (|devaluate| |d|) |$packages|)))))
```

———-

51.1.84 defun ?t

[isgenvar p876]
[get p**??**]
[sayMSG p329]

[bright p??]
[rassocSub p861]
[qcar p??]
[qcdr p??]
[isDomainOrPackage p865]
[isDomain p??]
[reportSpadTrace p882]
[take p??]
[sayBrightly p??]
[devaluate p??]
[$mapSubNameAlist p??]
[$InteractiveFrame p??]
[/tracenames p??]

— **defun ?t** —

```
(defun |?t| ()
 (let (llm d suffix l)
  (declare (special /tracenames |$InteractiveFrame| |$mapSubNameAlist|))
  (if (null /tracenames)
   (|sayMSG| (|bright| "nothing is traced"))
   (progn
    (dolist (x /tracenames)
     (cond
      ((and (atom x) (null (isgenvar x)))
       (progn
        (cond
         ((setq llm (|get| x '|localModemap| |$InteractiveFrame|))
           (setq x (list (cadar llm)))))
        (|sayMSG|
         `("Function" ,@(|bright| (|rassocSub| x |$mapSubNameAlist|))
           "traced"))))))
    (dolist (x /tracenames)
     (cond
      ((and (consp x)
            (progn (setq d (qcar x)) (setq l (qcdr x)) t)
            (|isDomainOrPackage| d))
       (progn
        (setq suffix (cond ((|isDomain| d) "domain") (t "package")))
        (|sayBrightly|
         `("   Functions traced in " ,suffix |%b| ,(|devaluate| d) |%d| ":"))
        (dolist (x (|orderBySlotNumber| l))
          (|reportSpadTrace| '|   | (TAKE 4 x)))
        (terpri)))))))))
```

———-

51.1.85 defun tracelet

[gensymp p??]
[stupidIsSpadFunction p896]
[bpiname p??]
[lassoc p??]
[union p??]
[setletprintflag p??]
[isgenvar p876]
[compileBoot p897]
[delete p??]
[$traceletflag p??]
[$QuickLet p??]
[$letAssoc p??]
[$traceletFunctions p??]

— **defun tracelet** —

```
(defun |tracelet| (fn |vars|)
 (prog ($traceletflag |$QuickLet| l)
  (declare (special $traceletflag |$QuickLet| |$letAssoc|
                    |$traceletFunctions|))
  (return
   (progn
    (cond
     ((and (gensymp fn) (|stupidIsSpadFunction| (eval fn)))
       (setq fn (eval fn))
       (cond
        ((compiled-function-p fn) (setq fn (bpiname fn)))
        (t nil))))
    (cond
     ((eq fn '|Undef|) nil)
     (t
       (setq |vars|
         (cond
          ((eq |vars| '|all|) '|all|)
          ((setq l (lassoc fn |$letAssoc|)) (|union| |vars| l))
          (t |vars|)))
       (setq |$letAssoc| (cons (cons fn |vars|) |$letAssoc|))
       (cond (|$letAssoc| (setletprintflag t)))
       (setq $traceletflag t)
       (setq |$QuickLet| nil)
       (cond
        ((and (null (member fn |$traceletFunctions|))
              (null (isgenvar fn))
              (compiled-function-p (symbol-function fn))
              (null (|stupidIsSpadFunction| fn))
              (null (gensymp fn)))
```

```
          (progn
           (setq |$traceletFunctions| (cons fn |$traceletFunctions|))
           (|compileBoot| fn)
           (setq |$traceletFunctions|
             (|delete| fn |$traceletFunctions|))))))))))))
```

———-

51.1.86 defun breaklet

[gensymp p??]
[stupidIsSpadFunction p896]
[bpiname p??]
[lassoc p??]
[assoc p??]
[union p??]
[setletprintflag p??]
[compileBoot p897]
[delete p??]
[$QuickLet p??]
[$letAssoc p??]
[$traceletFunctions p??]

— **defun breaklet** —

```
(defun |breaklet| (fn |vars|)
 (prog (|$QuickLet| |fnEntry| |pair|)
  (declare (special |$QuickLet| |$letAssoc| |$traceletFunctions|))
  (return
   (progn
    (cond
     ((and (gensymp fn) (|stupidIsSpadFunction| (eval fn)))
       (setq fn (eval fn))
       (cond
        ((compiled-function-p fn) (setq fn (bpiname fn)))
        (t nil))))
    (cond
     ((eq fn '|Undef|) nil)
     (t
      (setq |fnEntry| (lassoc fn |$letAssoc|))
      (setq |vars|
       (cond
        ((setq |pair| (|assoc| 'break |fnEntry|))
          (|union| |vars| (cdr |pair|)))
        (t |vars|)))
      (setq |$letAssoc|
       (cond
```

```
        ((null |fnEntry|)
          (cons (cons fn (list (cons 'break |vars|))) |$letAssoc|))
        (|pair| (rplacd |pair| |vars|) |$letAssoc|)))
      (cond (|$letAssoc| (setletprintflag t)))
      (setq |$QuickLet| nil)
      (cond
       ((and (null (member fn |$traceletFunctions|))
             (null (|stupidIsSpadFunction| fn))
             (null (gensymp fn)))
        (progn
         (setq |$traceletFunctions| (cons fn |$traceletFunctions|))
         (|compileBoot| fn)
         (setq |$traceletFunctions|
          (|delete| fn |$traceletFunctions|)))))))))
```

———-

51.1.87 defun stupidIsSpadFunction

[strpos p1039]
[pname p1039]

— **defun stupidIsSpadFunction** —

```
(defun |stupidIsSpadFunction| (fn)
 (strpos ";" (pname fn) 0 nil))
```

———-

51.1.88 defun break

[MONITOR,EVALTRAN p**??**]
[enable-backtrace p**??**]
[sayBrightly p**??**]
[interrupt p**??**]
[/breakcondition p**??**]

— **defun break** —

```
(defun |break| (msg)
 (prog (condition)
  (declare (special /breakcondition))
  (return
   (progn
    (setq condition (|MONITOR,EVALTRAN| /breakcondition nil))
```

```
    (when (eval condition)
      (|sayBrightly| msg)
      (interrupt))))))
```

———-

51.1.89 defun compileBoot

[/D,1 p??]

— **defun compileBoot** —

```
(defun |compileBoot| (fn)
 (|/D,1| (list fn) '(/comp) nil nil))
```

———-

Chapter 52

)undo help page Command

52.1 undo help page man page

— undo.help —

```
========================================================================
A.27.  )undo
========================================================================

User Level Required:  interpreter

Command Syntax:

  - )undo
  - )undo integer
  - )undo integer [option]
  - )undo )redo

where option is one of

  - )after
  - )before

Command Description:

This command is used to restore the state of the user environment to an
earlier point in the interactive session. The argument of an )undo is an
integer which must designate some step number in the interactive session.

)undo n
)undo n )after
```

```
These commands return the state of the interactive environment to that
immediately after step n. If n is a positive number, then n refers to step
nummber n. If n is a negative number, it refers to the nth previous command
(that is, undoes the effects of the last -n commands).

A )clear all resets the )undo facility. Otherwise, an )undo undoes the effect
of )clear with options properties, value, and mode, and that of a previous
undo. If any such system commands are given between steps n and n + 1 (n >
0), their effect is undone for )undo m for any 0 < m <= n .

The command )undo is equivalent to )undo -1 (it undoes the effect of the
previous user expression). The command )undo 0 undoes any of the above system
commands issued since the last user expression.

)undo n )before

This command returns the state of the interactive environment to that
immediately before step n. Any )undo or )clear system commands given before
step n will not be undone.

)undo )redo

This command reads the file redo.input. created by the last )undo command.
This file consists of all user input lines, excluding those backtracked over
due to a previous )undo.

The command )history )write will eliminate the ‘‘undone’’ command lines of
your program.

Also See:
o )history
```

———-

1

52.2 Evaluation

Some Antique Comments About the Interpreter

EVAL BOOT contains the top level interface to the Scratchhpad-II interpreter. The Entry point into the interpreter from the parser is processInteractive.

The type analysis algorithm is contained in the file BOTMUP BOOT, and MODSEL boot, the map handling routines are in MAP BOOT and NEWMAP BOOT, and the interactive coerce routines are in COERCE BOOT and COERCEFN BOOT.

[1] “history” (35.4.7 p 578)

Conventions: All spad values in the interpreter are passed around in triples. These are lists of three items:

```
[value,mode,environment]
```

The value may be wrapped (this is a pair whose CAR is the atom WRAPPED and whose CDR is the value), which indicates that it is a real value, or unwrapped in which case it needs to be EVALed to produce the proper value. The mode is the type of value, and should always be completely specified (not contain `$EmptyMode`). The environment is always empty, and is included for historical reasons.

Modemaps: Modemaps are descriptions of compiled Spad function which the interpreter uses to perform type analysis. They consist of patterns of types for the arguments, and conditions the types must satisfy for the function to apply. For each function name there is a list of modemaps in file modemap DATABASE for each distinct function with that name. The following is the list of the modemaps for "*" (multiplication. The first modemap (the one with the labels) is for module mltiplication which is multiplication of an element of a module by a member of its scalar domain.

This is the signature pattern for the modemap, it is of the form:

```
    (DomainOfComputation TargetType <ArgumentType ...>)
          |
          |                This is the predicate that needs to be
          |                 satisfied for the modemap to apply
          |                            |
          V                            |
     /-----------/                     |
( ( (*1 *1 *2 *1)                      V
      /-------------------------------------------------------------/
    ( (AND (ofCategory *1 (Module *2)) (ofCategory *2 (SimpleRing))) )
     . CATDEF) <-- This is the file where the function was defined
  ( (*1 *1 *2 *1)
    ( (AND (isDomain *2 (Integer)) (ofCategory *1 (AbelianGroup))) )
     . CATDEF)
  ( (*1 *1 *2 *1)
    ( (AND
        (isDomain *2 (NonNegativeInteger))
        (ofCategory *1 (AbelianMonoid))) )
     . CATDEF)
  ((*1 *1 *1 *1) ((ofCategory *1 (SemiGroup)) ) . CATDEF)
     )
```

Environments: Environments associate properties with atoms.

Some common properties are:

- **modeSet:** During interpretation we build a modeSet property for each node in the expression. This is (in theory) a list of all the types possible for the node. In the current implementation these modeSets always contain a single type.

- **value:** Value properties are always triples. This is where the values of variables are stored. We also build value properties for internal nodes during the bottom up phase.
- **mode:** This is the declared type of an identifier.

There are several different environments used in the interpreter:

- **`$InteractiveFrame`:** this is the environment where the user values are stored. Any side effects of evaluation of a top-level expression are stored in this environment. It is always used as the starting environment for interpretation.
- **$e:** This is the name used for `$InteractiveFrame` while interpreting.
- **`$env`:** This is local environment used by the interpreter. Only temporary information (such as types of local variables is stored in `$env`. It is thrown away after evaluation of each expression.

Frequently used global variables:

- `$genValue`: if true then evaluate generated code, otherwise leave code unevaluated. If `$genValue` is false then we are compiling.
- `$op`: name of the top level operator (unused except in map printing)
- `$mapList`: list of maps being type analyzed, used in recursive map type anlysis.
- `$compilingMap`: true when compiling a map, used to detect where to THROW when interpret-only is invoked
- `$compilingLoop`: true when compiling a loop body, used to control nesting level of interp-only loop CATCH points
- `$interpOnly`: true when in interpret only mode, used to call alternate forms of COLLECT and REPEAT.
- `$inCOLLECT`: true when compiling a COLLECT, used only for hacked stream compiler.
- `$StreamFrame`: used in printing streams, it is the environment where local stream variables are stored
- `$declaredMode`: Weak type propagation for symbols, set in upCOERCE and upLET. This variable is used to determine the alternate polynomial types of Symbols.
- `$localVars`: list of local variables in a map body
- `$MapArgumentTypeList`: hack for stream compilation

52.2.1 defun evalDomain

[sayMSG p329]
[concat p1041]
[prefix2String p??]
[startTimingProcess p??]
[eval p??]
[mkEvalable p903]
[stopTimingProcess p??]
[$evalDomain p903]

— **defun evalDomain** —

```
(defun |evalDomain| (form)
 (let (result)
 (declare (special |$evalDomain|))
  (when |$evalDomain|
    (|sayMSG|
      (|concat| "   instantiating" '|%b| (|prefix2String| form) '|%d|)))
  (|startTimingProcess| '|instantiation|)
  (setq result (|eval| (|mkEvalable| form)))
  (|stopTimingProcess| '|instantiation|)
  result))
```

———-

52.2.2 defun mkEvalable

[qcar p??]
[qcdr p??]
[mkEvalable p903]
[devaluate p??]
[mkEvalableRecord p905]
[mkEvalableUnion p905]
[mkEvalableMapping p905]
[loadIfNecessary p??]
[getdatabase p1001]
[mkq p??]
[constructor? p??]
[fbpip p??]
[bpiname p??]
[$Integer p??]
[$EmptyMode p??]

— **defun mkEvalable** —

```
(defun |mkEvalable| (form)
 (let (op argl kind cosig)
 (declare (special |$Integer| |$EmptyMode|))
  (cond
   ((consp form)
     (setq op (qcar form))
     (setq argl (qcdr form))
     (cond
      ((eq op 'quote) form)
      ((eq op 'wrapped) (|mkEvalable| (|devaluate| argl)))
      ((eq op '|Record|) (|mkEvalableRecord| form))
      ((eq op '|Union|) (|mkEvalableUnion| form))
      ((eq op '|Mapping|) (|mkEvalableMapping| form))
      ((eq op '|Enumeration|) form)
      (t
       (|loadIfNecessary| op)
       (setq kind (getdatabase op 'constructorkind))
       (cond
         ((setq cosig (getdatabase op 'cosig))
          (cons op
           (loop for x in argl for typeFlag in (rest cosig)
            collect
             (cond
              (typeFlag
               (cond
                ((eq kind '|category|) (mkq x))
                ((vecp x) (mkq x))
                (t
                  (|loadIfNecessary| x)
                  (|mkEvalable| x))))
              ((and (consp x) (eq (qcar x) 'quote)) x)
              ((and (consp x) (eq (qcar x) '|#|) (consp (qcdr x))
                    (eq (qcdr (qcdr x)) nil))
                (list 'size (mkq (qcar (qcdr x)))))
              (t (mkq x))))))
         (t
          (cons op
           (loop for x in argl
            collect (|mkEvalable| x))))))))
     ((equal form |$EmptyMode|) |$Integer|)
     ((and (identp form) (|constructor?| form)) (list form))
     ((fbpip form) (bpiname form))
     (t form))))
```

———-

52.2.3 defun mkEvalableUnion

[mkEvalable p903]

— **defun mkEvalableUnion** —

```
(defun |mkEvalableUnion| (form)
  (cond
   ((|isTaggedUnion| form)
    (cons
     (car form)
     (loop for item in (rest form)
      collect (list '|:| (second item) (|mkEvalable| (third item))))))
   (t
    (cons (car form)
     (loop for d in (rest form)
      collect (|mkEvalable| d))))))
```

———-

52.2.4 defun mkEvalableRecord

[mkEvalable p903]

— **defun mkEvalableRecord** —

```
(defun |mkEvalableRecord| (form)
  (cons
   (car form)
   (loop for item in (rest form)
    collect (list (quote |:|) (second item) (|mkEvalable| (third item))))))
```

———-

52.2.5 defun mkEvalableMapping

[mkEvalable p903]

— **defun mkEvalableMapping** —

```
(defun |mkEvalableMapping| (form)
 (cons
  (car form)
  (loop for d in (rest form)
   collect (|mkEvalable| d))))
```

———-

52.2.6 defun evaluateType

Takes a parsed, unabbreviated type and evaluates it, replacing type valued variables with their values, and calling bottomUp on non-type valued arguemnts to the constructor and finally checking to see whether the type satisfies the conditions of its modemap [isDomainValuedVariable p949]
[qcar p**??**]
[qcdr p**??**]
[mkAtree p**??**]
[bottomUp p**??**]
[objVal p**??**]
[getValue p**??**]
[evaluateSignature p910]
[member p1042]
[evaluateType p906]
[constructor? p**??**]
[throwEvalTypeMsg p909]
[$EmptyMode p**??**]
[$expandSegments p**??**]

— defun evaluateType —

```
(defun |evaluateType| (form)
 (let (|$expandSegments| domain formp op argl)
  (declare (special |$expandSegments| |$EmptyMode|))
  (cond
    ((setq domain (|isDomainValuedVariable| form)) domain)
    ((equal form |$EmptyMode|) form)
    ((eq form '?) |$EmptyMode|)
    ((stringp form) form)
    ((eq form '$) form)
    (t
     (setq |$expandSegments| nil)
     (cond
       ((and (consp form) (eq (qcar form) '|typeOf|) (consp (qcdr form))
             (eq (qcdr (qcdr form)) nil))
         (setq formp (|mkAtree| form))
         (|bottomUp| formp)
         (|objVal| (|getValue| formp)))
       ((consp form)
         (setq op (qcar form))
         (setq argl (qcdr form))
         (cond
          ((eq op 'category)
            (cond
```

```
        ((consp argl)
          (cons op
           (cons (qcar argl)
            (loop for s in (qcdr argl)
             collect (|evaluateSignature| s)))))
        (t form)))
      ((|member| op '(|Join| |Mapping|))
        (cons op
         (loop for arg in argl
          collect (|evaluateType| arg))))
      ((eq op '|Union|)
        (cond
         ((and argl (consp (car argl)) (consp (qcdr (car argl)))
               (consp (qcdr (qcdr (car argl))))
               (eq (qcdr (qcdr (qcdr (car argl)))) nil)
               (|member| (qcar (car argl)) '(|:| |Declare|)))
          (cons op
           (loop for item in argl
            collect
             (list '|:| (second item) (|evaluateType| (third item))))))
         (t
          (cons op
           (loop for arg in argl
            collect (|evaluateType| arg))))))
      ((eq op '|Record|)
        (cons op
         (loop for item in argl
          collect
           (list '|:| (second item) (|evaluateType| (third item))))))
      ((eq op '|Enumeration|) form)
      (t (|evaluateType1| form))))
   ((|constructor?| form)
     (if (atom form)
       (|evaluateType| (list form))
       (|throwEvalTypeMsg| 'S2IE0003 (list form form))))
   (t (|throwEvalTypeMsg| 'S2IE0004 (list form)))))))
```

———-

52.2.7 defun Eval args passed to a constructor

Evaluates the arguments passed to a constructor [constructor? p**??**]
[getConstructorSignature p**??**]
[throwEvalTypeMsg p909]
[replaceSharps p948]
[categoryForm? p**??**]
[evaluateType p906]

[evalCategory p949]
[getdatabase p1001]
[mkAtree p??]
[putTarget p??]
[bottumUp p??]
[qcar p??]
[qcdr p??]
[getAndEvalConstructorArgument p948]
[coerceOrRetract p??]
[objValUnwrap p??]
[throwKeyedMsgCannotCoerceWithValue p??]
[makeOrdinal p910]
[$quadSymbol p??]
[$EmptyMode p??]

— defun evaluateType1 —

```
(defun |evaluateType1| (form)
 (let (op argl sig ml xp tree tmp1 m1 z1 zt zv v typeList (argnum 0))
 (declare (special |$quadSymbol| |$EmptyMode|))
  (setq op (car form))
  (setq argl (cdr form))
  (cond
   ((|constructor?| op)
     (cond
      ((null (setq sig (|getConstructorSignature| form)))
       (|throwEvalTypeMsg| 'S2IE0005 (list form)))
      (t
       (setq ml (cdr sig))
       (setq ml (|replaceSharps| ml form))
       (cond
        ((not (eql (|#| argl) (|#| ml)))
          (|throwEvalTypeMsg| 'S2IE0003 (list form form)))
        (t
         (loop for x in argl for m in ml
         do
          (setq typeList
           (cons
            (cond
             ((|categoryForm?| m)
               (setq m (|evaluateType| (subst x '$ m)))
               (if (|evalCategory| (setq xp (|evaluateType| x)) m)
                  xp
                  (|throwEvalTypeMsg| 'S2IE0004 (list form))))
             (t
              (setq m (|evaluateType| m))
              (cond
               ((and (eq (getdatabase (|opOf| m) 'constructorkind) '|domain|)
```

```
                    (setq tree (|mkAtree| x))
                    (|putTarget| tree m)
                    (progn
                     (setq tmp1 (|bottomUp| tree))
                     (and (consp tmp1)
                          (eq (qcdr tmp1) nil))))
                  (setq m1 (qcar tmp1))
                  (setq z1 (|getAndEvalConstructorArgument| tree))
                  (setq zt (car z1))
                  (setq zv (cdr z1))
                  (if (setq v (|coerceOrRetract| z1 m))
                    (|objValUnwrap| v)
                    (|throwKeyedMsgCannotCoerceWithValue| zv zt m)))
                 (t
                  (when (equal x |$EmptyMode|) (setq x |$quadSymbol|))
                  (|throwEvalTypeMsg| 'S2IE0006
                   (list (|makeOrdinal| (incf argnum)) m form))))))
             typeList)))
      (cons op (nreverse typeList)))))))
   (t (|throwEvalTypeMsg| 'S2IE0007 (list op))))))
```

———-

52.2.8 defvar $noEvalTypeMsg

— **initvars** —

```
(defvar |$noEvalTypeMsg| nil)
```

———-

52.2.9 defun throwEvalTypeMsg

[spadThrow p**??**]
[throwKeyedMsg p**??**]
[$noEvalTypeMsg p909]

— **defun throwEvalTypeMsg** —

```
(defun |throwEvalTypeMsg| (msg args)
 (declare (special |$noEvalTypeMsg|))
 (if |$noEvalTypeMsg|
  (|spadThrow|)
  (|throwKeyedMsg| msg args)))
```

———-

52.2.10 defun makeOrdinal

— **defun makeOrdinal** —

```
(defun |makeOrdinal| (i)
 (elt '(|first| |second| |third| |fourth| |fifth| |sixth| |seventh|
        |eighth| |ninth| |tenth|)
  (1- i)))
```

———-

52.2.11 defun evaluateSignature

Calls evaluateType on a signature [evaluateType p906]

— **defun evaluateSignature** —

```
(defun |evaluateSignature| (sig)
  (cond
   ((and (consp sig) (eq (qcar sig) 'signature) (consp (qcdr sig))
         (consp (qcdr (qcdr sig))) (eq (qcdr (qcdr (qcdr sig))) nil))
    (cons 'signature (cons (qcar (qcdr sig))
      (list
       (loop for z in (qcar (qcdr (qcdr sig)))
        collect (if (eq z '$) z (|evaluateType| z)))))))
   (t sig)))
```

———-

52.3 Data Structures

`$frameRecord = [delta1, delta2,... ]` where delta(i) contains changes in the "backwards" direction. Each delta(i) has the form `((var . proplist)...)` where proplist denotes an ordinary proplist. For example, an entry of the form `((x (value) (mode (Integer)))...)` indicates that to undo 1 step, x's value is cleared and its mode should be set to (Integer).

A delta(i) of the form (systemCommand . delta) is a special delta indicating changes due to system commands executed between the last command and the current command. By recording these deltas separately, it is possible to undo to either BEFORE or AFTER the command. These special delta(i)s are given ONLY when a a system command is given which alters the environment.

Note: recordFrame('system) is called before a command is executed, and recordFrame('normal) is called after (see processInteractive1). If no changes are found for former, no special entry is given.

The `$previousBindings` is a copy of the `CAAR $InteractiveFrame`. This is used to compute the delta(i)s stored in `$frameRecord`.

52.4 Functions

52.4.1 Initial Undo Variables

```
$undoFlag := true     --Default setting for undo is "on"
$frameRecord  := nil  --Initial setting for frame record
$previousBindings := nil
```

52.4.2 defvar $undoFlag

— **initvars** —

```
(defvar |$undoFlag| t "t means we record undo information")
```

————-

52.4.3 defvar $frameRecord

— **initvars** —

```
(defvar |$frameRecord| nil "a list of value changes")
```

————-

52.4.4 defvar $previousBindings

— **initvars** —

```
(defvar |$previousBindings| nil "a copy of Interactive Frame info for undo")
```

————-

52.4.5 defvar $reportUndo

— **initvars** —

```
(defvar |$reportUndo| nil "t means we report the steps undo takes")
```

————-

52.4.6 defun undo

[stringPrefix? p??]
[pname p1039]
[read p638]
[userError p??]
[qcdr p??]
[qcar p??]
[spaddifference p??]
[identp p1040]
[undoSteps p920]
[undoCount p919]
[$options p??]
[$InteractiveFrame p??]

— **defun undo** —

```
(defun |undo| (l)
 (let (tmp1 key s undoWhen n)
 (declare (special |$options| |$InteractiveFrame|))
  (setq undoWhen '|after|)
  (when
    (and (consp |$options|)
         (eq (qcdr |$options|) nil)
         (progn
          (setq tmp1 (qcar |$options|))
          (and (consp tmp1)
               (eq (qcdr tmp1) nil)
               (progn (setq key (qcar tmp1)) t)))
     (cond
      ((|stringPrefix?| (setq s (pname key)) "redo")
        (setq |$options| nil)
        (|read| '(|redo.input|)))
      ((null (|stringPrefix?| s "before"))
        (|userError| "only option to undo is \")redo\""))
      (t
        (setq undoWhen '|before|)))))
```

```
  (if (null l)
    (setq n (spaddifference 1))
    (setq n (car l)))
  (when (identp n)
    (setq n (parse-integer (pname n)))
    (unless (integerp n)
      (|userError| "undo argument must be an integer")))
  (setq |$InteractiveFrame| (|undoSteps| (|undoCount| n) undoWhen))
  nil))
```

———-

52.4.7 defun recordFrame

[kar p??]
[diffAlist p914]
[seq p??]
[exit p??]
[$undoFlag p911]
[$frameRecord p911]
[$InteractiveFrame p??]
[$previousBindings p911]

— **defun recordFrame** —

```
(defun |recordFrame| (systemNormal)
 (prog (currentAlist delta)
 (declare (special |$undoFlag| |$frameRecord| |$InteractiveFrame|
    |$previousBindings|))
  (return
   (seq
    (cond
     ((null |$undoFlag|) nil)
     (t
       (setq currentAlist (kar |$frameRecord|))
       (setq delta
         (|diffAlist| (caar |$InteractiveFrame|) |$previousBindings|))
       (cond
        ((eq systemNormal '|system|)
          (cond
           ((null delta)
             (return nil))
           (t
             (setq delta (cons '|systemCommand| delta))))))
       (setq |$frameRecord| (cons delta |$frameRecord|))
       (setq |$previousBindings|
        (prog (tmp0)
```

```
        (setq tmp0 nil)
        (return
         (do ((tmp1 (caar |$InteractiveFrame|) (cdr tmp1)) (x nil))
             ((or (atom tmp1)
                  (progn (setq x (car tmp1)) nil))
              (nreverse0 tmp0))
          (seq
           (exit
            (setq tmp0
             (cons
              (cons
               (car x)
               (prog (tmp2)
                (setq tmp2 nil)
                (return
                 (do ((tmp3 (cdr x) (cdr tmp3)) (y nil))
                     ((or (atom tmp3)
                          (progn (setq y (car tmp3)) nil))
                      (nreverse0 tmp2))
                  (seq
                   (exit
                    (setq tmp2 (cons (cons (car y) (cdr y)) tmp2)))))))))
             tmp0)))))))
     (car |$frameRecord|)))))))
```

52.4.8 defun diffAlist

```
diffAlist(new,old) ==
--record only those properties which are different
  for (pair := [name,:proplist]) in new repeat
    -- name has an entry both in new and old world
    -- (1) if the old world had no proplist for that variable, then
    --     record NIL as the value of each new property
    -- (2) if the old world does have a proplist for that variable, then
    --     a) for each property with a value: give the old value
    --     b) for each property missing:      give NIL as the old value
    oldPair := ASSQ(name,old) =>
      null (oldProplist := CDR oldPair) =>
      --record old values of new properties as NIL
        acc := [ [name,:[ [prop] for [prop,:.] in proplist] ],:acc]
      deltas := nil
      for (propval := [prop,:val]) in proplist repeat
        null (oldPropval := ASSOC(prop,oldProplist)) => --missing property
          deltas := [ [prop],:deltas]
        EQ(CDR oldPropval,val) => 'skip
        deltas := [oldPropval,:deltas]
```

```
      deltas => acc := [ [name,:NREVERSE deltas],:acc]
    acc := [ [name,:[ [prop] for [prop,:.] in proplist] ],:acc]
--record properties absent on new list (say, from a )cl all)
  for (oldPair := [name,:r]) in old repeat
    r and null LASSQ(name,new) =>
      acc := [oldPair,:acc]
    -- name has an entry both in new and old world
    -- (1) if the new world has no proplist for that variable
    --     (a) if the old world does, record the old proplist
    --     (b) if the old world does not, record nothing
    -- (2) if the new world has a proplist for that variable, it has
    --     been handled by the first loop.
  res := NREVERSE acc
  if BOUNDP '$reportUndo and $reportUndo then reportUndo res
  res
```

[assq p1044]
[tmp1 p??]
[seq p??]
[exit p??]
[assoc p??]
[lassq p??]
[reportUndo p917]

— **defun diffAlist** —

```
(defun |diffAlist| (new old)
 (prog (proplist oldPair oldProplist val oldPropval deltas prop name r acc res)
  (return
   (seq
    (progn
     (do ((tmp0 new (cdr tmp0)) (pair nil))
         ((or (atom tmp0)
              (progn (setq pair (car tmp0)) nil)
              (progn
               (progn
                (setq name (car pair))
                (setq proplist (cdr pair))
                pair)
               nil))
             nil)
      (seq
       (exit
        (cond
         ((setq oldPair (assq name old))
          (cond
           ((null (setq oldProplist (cdr oldPair)))
             (setq acc
              (cons
```

```
      (cons
        name
        (prog (tmp1)
         (setq tmp1 nil)
         (return
          (do ((tmp2 proplist (cdr tmp2)) (tmp3 nil))
              ((or (atom tmp2)
                   (progn (setq tmp3 (car tmp2)) nil)
                   (progn
                    (progn (setq prop (car tmp3)) tmp3)
                    nil))
                (nreverse0 tmp1))
           (seq
            (exit
             (setq tmp1 (cons (cons prop nil) tmp1))))))))
      acc)))
  (t
    (setq deltas nil)
    (do ((tmp4 proplist (cdr tmp4)) (|propval| nil))
        ((or (atom tmp4)
             (progn (setq |propval| (car tmp4)) nil)
             (progn
              (progn
               (setq prop (car |propval|))
               (setq val (cdr |propval|))
               |propval|)
              nil))
          nil)
     (seq
      (exit
       (cond
        ((null (setq oldPropval (|assoc| prop oldProplist)))
          (setq deltas (cons (cons prop nil) deltas)))
        ((eq (cdr oldPropval) val) '|skip|)
        (t (setq deltas (cons oldPropval deltas)))))))
    (when deltas
      (setq acc
       (cons (cons name (nreverse deltas)) acc))))))
(t
  (setq acc
   (cons
    (cons
     name
     (prog (tmp5)
      (setq tmp5 nil)
      (return
       (do ((tmp6 proplist (cdr tmp6)) (tmp7 nil))
           ((or (atom tmp6)
                (progn (setq tmp7 (CAR tmp6)) nil)
                (progn
```

```
                        (progn (setq prop (CAR tmp7)) tmp7)
                        nil))
                    (nreverse0 tmp5))
               (seq
                (exit
                 (setq tmp5 (cons (cons prop nil) tmp5))))))))
          acc)))))))
    (seq
     (do ((tmp8 old (cdr tmp8)) (oldPair nil))
         ((or (atom tmp8)
              (progn (setq oldPair (car tmp8)) nil)
              (progn
               (progn
                (setq name (car oldPair))
                (setq r (cdr oldPair))
                oldPair)
               nil))
          nil)
       (seq
        (exit
         (cond
          ((and r (null (lassq name new)))
           (exit
            (setq acc (cons oldPair acc))))))))
     (setq res (nreverse acc))
     (cond
      ((and (boundp '|$reportUndo|) |$reportUndo|)
        (|reportUndo| res)))
     (exit res)))))
```

———-

52.4.9 defun reportUndo

This function is enabled by setting `$reportUndo` to a non-nil value. An example of the output generated is:

```
r := binary(22/7)


          ---
   (1)  11.001
                                                    Type: BinaryExpansion
Properties of % ::
  value was: NIL
  value is:  ((|BinaryExpansion|) WRAPPED . #(1 (1 1) NIL (0 0 1)))
Properties of r ::
  value was: NIL
```

```
  value is:  ((|BinaryExpansion|) WRAPPED . #(1 (1 1) NIL (0 0 1)))
```

[seq p**??**]
[exit p**??**]
[sayBrightly p**??**]
[concat p1041]
[pname p1039]
[lassoc p**??**]
[sayBrightlyNT p**??**]
[pp p**??**]
[$InteractiveFrame p**??**]

— defun reportUndo —

```
(defun |reportUndo| (acc)
 (prog (name proplist curproplist prop value)
 (declare (special |$InteractiveFrame|))
  (return
   (seq
    (do ((tmp0 acc (cdr tmp0)) (tmp1 nil))
        ((or (atom tmp0)
             (progn (setq tmp1 (car tmp0)) nil)
             (progn
              (progn
               (setq name (car tmp1))
               (setq proplist (cdr tmp1))
               tmp1)
              nil))
            nil)
     (seq
      (exit
       (progn
        (|sayBrightly|
          (concat '|Properties of | (pname name) " ::"))
        (setq curproplist (lassoc name (caar |$InteractiveFrame|)))
        (do ((tmp2 proplist (cdr tmp2)) (tmp3 nil))
            ((or (atom tmp2)
                 (progn (setq tmp3 (car tmp2)) nil)
                 (progn
                  (progn
                   (setq prop (car tmp3))
                   (setq value (cdr tmp3))
                   tmp3)
                  nil))
                nil)
         (seq
          (exit
           (progn
```

```
(|sayBrightlyNT|
 (cons "  " (cons prop (cons " was: " nil))))
(|pp| value)
(|sayBrightlyNT|
 (cons "  " (cons prop (cons " is:  " nil))))
(|pp| (lassoc prop curproplist)))))))))))))))
```

———-

52.4.10 defun clearFrame

[clearCmdAll p499]
[$frameRecord p911]
[$previousBindings p911]

— **defun clearFrame** —

```
(defun |clearFrame| ()
 (declare (special |$frameRecord| |$previousBindings|))
  (|clearCmdAll|)
  (setq |$frameRecord| nil)
  (setq |$previousBindings| nil))
```

———-

52.4.11 Undo previous n commands

[spaddifference p**??**]
[userError p**??**]
[concat p1041]
[$IOindex p**??**]

— **defun undoCount** —

```
(defun |undoCount| (n)
 "Undo previous n commands"
 (prog (m)
 (declare (special |$IOindex|))
  (return
   (progn
    (setq m
     (cond
       ((>= n 0) (spaddifference (spaddifference |$IOindex| n) 1))
       (t (spaddifference n))))
    (cond
```

```
     ((>= m |$IOindex|)
       (|userError|
        (concat "Magnitude of undo argument must be less than step number ("
           (princ-to-string |$IOindex|) ").")))
     (t m))))))
```

———-

52.4.12 defun undoSteps

```
-- undoes m previous commands; if )before option, then undo one extra at end
--Example: if $IOindex now is 6 and m = 2 then general layout of $frameRecord,
--  after the call to recordFrame below will be:
--  (<change for systemcommands>
--  (<change for #5> <change for system commands>
--  (<change for #4> <change for system commands>
--  (<change for #3> <change for system commands>
--   <change for #2> <change for system commands>
--   <change for #1> <change for system commands>) where system
--  command entries are optional and identified by (systemCommand . change).
--  For a ")undo 3 )after", m = 2 and undoStep swill restore the environment
--  up to, but not including <change for #3>.
--  An "undo 3 )before" will additionally restore <change for #3>.
--  Thus, the later requires one extra undo at the end.
```

[writeInputLines p583]
[spaddifference p??]
[recordFrame p913]
[copy p??]
[undoSingleStep p921]
[qcdr p??]
[qcar p??]
[$IOindex p??]
[$InteractiveFrame p??]
[$frameRecord p911]

— **defun undoSteps** —

```
(defun |undoSteps| (m beforeOrAfter)
 (let (tmp1 tmp2 systemDelta lastTailSeen env)
 (declare (special |$IOindex| |$InteractiveFrame| |$frameRecord|))
  (|writeInputLines| '|redo| (spaddifference |$IOindex| m))
  (|recordFrame| '|normal|)
  (setq env (copy (caar |$InteractiveFrame|)))
  (do ((i 0 (1+ i)) (framelist |$frameRecord| (cdr framelist)))
      ((or (> i m) (atom framelist)) nil)
```

```
    (setq env (|undoSingleStep| (CAR framelist) env))
    (if (and (consp framelist)
             (progn
              (setq tmp1 (qcdr framelist))
              (and (consp tmp1)
                   (progn
                    (setq tmp2 (qcar tmp1))
                    (and (consp tmp2)
                         (eq (qcar tmp2) '|systemCommand|)
                         (progn
                          (setq systemDelta (qcdr tmp2))
                          t))))))
        (progn
          (setq framelist (cdr framelist))
          (setq env (|undoSingleStep| systemDelta env)))
        (setq lastTailSeen framelist)))
  (cond
   ((eq beforeOrAfter '|before|)
     (setq env (|undoSingleStep| (car (cdr lastTailSeen)) env))))
  (setq |$frameRecord| (cdr |$frameRecord|))
  (setq |$InteractiveFrame| (list (list env)))))
```

———-

52.4.13 defun undoSingleStep

```
undoSingleStep(changes,env) ==
--Each change is a name-proplist pair. For each change:
--  (1) if there exists a proplist in env, then for each prop-value change:
--      (a) if the prop exists in env, RPLAC in the change value
--      (b) otherwise, CONS it onto the front of prop-values for that name
--  (2) add change to the front of env
--  pp '"----Undoing 1 step--------"
--  pp changes
```

[assq p1044]
[seq p**??**]
[exit p**??**]
[lassoc p**??**]
[undoLocalModemapHack p923]

— **defun undoSingleStep** —

```
(defun |undoSingleStep| (changes env)
 (prog (name changeList pairlist proplist prop value node)
  (return
   (seq
```

```
(progn
 (do ((tmp0 changes (cdr tmp0)) (|change| nil))
     ((or (atom tmp0)
          (progn (setq |change| (car tmp0)) nil)
          (progn
           (progn
            (setq name (car |change|))
            (setq changeList (cdr |change|))
            |change|)
           nil))
         nil)
  (seq
   (exit
    (progn
     (when (lassoc '|localModemap| changeList)
       (setq changeList (|undoLocalModemapHack| changeList)))
     (cond
      ((setq pairlist (assq name env))
        (cond
         ((setq proplist (cdr pairlist))
           (do ((tmp1 changeList (cdr tmp1)) (pair nil))
               ((or (atom tmp1)
                    (progn (setq pair (car tmp1)) nil)
                    (progn
                     (progn
                      (setq prop (car pair))
                      (setq value (cdr pair))
                      pair)
                      nil))
                    nil)
             (seq
              (exit
               (cond
                ((setq node (assq prop proplist))
                  (rplacd node value))
                (t
                  (rplacd proplist
                    (cons (car proplist) (cdr proplist)))
                  (rplaca proplist pair)))))))
          (t (rplacd pairlist changeList))))
      (t
        (setq env (cons |change| env))))))))
 env)))))
```

———-

52.4.14 defun undoLocalModemapHack

[seq p??]
[exit p??]

— **defun undoLocalModemapHack** —

```
(defun |undoLocalModemapHack| (changeList)
 (prog (name value)
  (return
   (seq
    (prog (tmp0)
     (setq tmp0 nil)
     (return
      (do ((tmp1 changeList (cdr tmp1)) (pair nil))
          ((or (atom tmp1)
               (progn (setq pair (car tmp1)) nil)
               (progn
                (progn
                 (setq name (car pair))
                 (setq value (cdr pair))
                 pair)
                nil))
              (nreverse0 tmp0))
       (seq
        (exit
         (cond
          ((cond
             ((eq name '|localModemap|) (cons name nil))
             (t pair))
           (setq tmp0
            (cons
             (cond
              ((eq name '|localModemap|) (cons name nil))
              (t pair)) tmp0)))))))))))
```

———-

52.4.15 Remove undo lines from history write

Removing undo lines from `)hist )write linelist` [stringPrefix? p??]
[seq p??]
[exit p??]
[trimString p??]
[substring p??]
[charPosition p??]
[maxindex p??]

[undoCount p919]
[spaddifference p??]
[concat p1041]
[$currentLine p??]
[$IOindex p??]

— defun removeUndoLines —

```
(defun |removeUndoLines| (u)
 "Remove undo lines from history write"
 (prog (xtra savedIOindex s s1 m s2 x code c n acc)
 (declare (special |$currentLine| |$IOindex|))
  (return
   (seq
    (progn
     (setq xtra
      (cond
       ((stringp |$currentLine|) (cons |$currentLine| nil))
       (t (reverse |$currentLine|))))
     (setq xtra
      (prog (tmp0)
       (setq tmp0 nil)
       (return
        (do ((tmp1 xtra (cdr tmp1)) (x nil))
            ((or (atom tmp1)
                 (progn (setq x (car tmp1)) nil))
             (nreverse0 tmp0))
         (seq
          (exit
           (cond
            ((null (|stringPrefix?| ")history" x))
              (setq tmp0 (cons x tmp0)))))))))))
     (setq u (append u xtra))
     (cond
      ((null
        (prog (tmp2)
         (setq tmp2 nil)
         (return
          (do ((tmp3 nil tmp2) (tmp4 u (cdr tmp4)) (x nil))
              ((or tmp3 (atom tmp4) (progn (setq x (car tmp4)) nil)) tmp2)
           (seq
            (exit
             (setq tmp2
               (or tmp2 (|stringPrefix?| ")undo" x))))))))) u)
      (t
        (setq savedIOindex |$IOindex|)
        (setq |$IOindex| 1)
        (do ((y u (cdr y)))
            ((atom y) nil)
```

```
 (seq
  (exit
   (cond
    ((eql (elt (setq x (car y)) 0) #\) )
      (cond
       ((|stringPrefix?| ")undo"
                         (setq s (|trimString| x)))
        (setq s1 (|trimString| (substring s 5 nil)))
        (cond
          ((not (string= s1 ")redo"))
            (setq m (|charPosition| #\) s1 0))
            (setq code
             (cond
               ((> (maxindex s1) m) (elt s1 (1+ m)))
               (t #\a)))
            (setq s2 (|trimString| (substring s1 0 m)))))
        (setq n
         (cond
          ((string= s1 ")redo")
             0)
          ((not (string= s2 ""))
            (|undoCount| (parse-integer s2)))
          (t -1)))
        (rplaca y
          (concat ">" code (princ-to-string n))))
       (t nil)))
    (t (setq |$IOindex| (1+ |$IOindex|)))))))
(setq acc nil)
(do ((y (nreverse u) (cdr y)))
    ((atom y) nil)
 (seq
  (exit
   (cond
    ((eql (elt (setq x (car y)) 0) #\>)
      (setq code (elt x 1))
      (setq n (parse-integer (substring x 2 nil)))
      (setq y (cdr y))
      (do ()
          ((null y) nil)
       (seq
        (exit
         (progn
          (setq c (car y))
          (cond
           ((or (eql (elt c 0) #\))
                (eql (elt c 0) #\>))
             (setq y (cdr y)))
           ((eql n 0)
             (return nil))
           (t
```

```
              (setq n (spaddifference n 1))
              (setq y (cdr y))))))))
      (cond
       ((and y (not (eql code #\b)))
         (setq acc (cons c acc)))))
    (t (setq acc (cons x acc)))))))
(setq |$IOindex| savedIOindex)
acc)))))))
```

———-

Chapter 53

)what help page Command

53.1 what help page man page

— what.help —

```
====================================================================
A.28.  )what
====================================================================

User Level Required:  interpreter

Command Syntax:

  - )what categories pattern1 [pattern2 ...]
  - )what commands   pattern1 [pattern2 ...]
  - )what domains    pattern1 [pattern2 ...]
  - )what operations pattern1 [pattern2 ...]
  - )what packages   pattern1 [pattern2 ...]
  - )what synonym    pattern1 [pattern2 ...]
  - )what things     pattern1 [pattern2 ...]
  - )apropos         pattern1 [pattern2 ...]

Command Description:

This command is used to display lists of things in the system. The patterns
are all strings and, if present, restrict the contents of the lists. Only
those items that contain one or more of the strings as substrings are
displayed. For example,

)what synonym

displays all command synonyms,
```

```
)what synonym ver

displays all command synonyms containing the substring ‘‘ver’’,

)what synonym ver pr

displays all command synonyms containing the substring ‘‘ver’’ or the
substring ‘‘pr’’. Output similar to the following will be displayed

---------------- System Command Synonyms -----------------


user-defined synonyms satisfying patterns:
      ver pr


  )apr ........................... )what things
  )apropos ....................... )what things
  )prompt ........................ )set message prompt
  )version ....................... )lisp *yearweek*

Several other things can be listed with the )what command:

  categories displays a list of category constructors.
  commands  displays a list of  system commands available  at your
    user-level. Your user-level is set via the )set userlevel command. To get
    a description of a particular command, such as ‘‘)what’’, issue )help
    what.
  domains   displays a list of domain constructors.
  operations displays a list of operations in  the system library.
    It is recommended that you qualify this command with one or more
    patterns, as there are thousands of operations available. For example,
    say you are looking for functions that involve computation of
    eigenvalues. To find their names, try )what operations eig. A rather
    large list of operations is loaded into the workspace when this command
    is first issued. This list will be deleted when you clear the workspace
    via )clear all or )clear completely. It will be re-created if it is
    needed again.
  packages  displays a list of package constructors.
  synonym  lists system command synonyms.
  things    displays all  of the  above types for  items containing
    the pattern strings as substrings. The command synonym )apropos is
    equivalent to )what things.

Also See:
o )display
o )set
o )show
```

————-

[1]

53.1.1 defvar $whatOptions

— **initvars** —

```
(defvar |$whatOptions| '(|operations| |categories| |domains| |packages|
                         |commands| |synonyms| |things|))
```

————-

53.1.2 defun what

[whatSpad2Cmd p930]

— **defun what** —

```
(defun |what| (l)
 (|whatSpad2Cmd| l))
```

————-

53.1.3 defun whatSpad2Cmd,fixpat

[qcar p**??**]
[downcase p**??**]

— **defun whatSpad2Cmd,fixpat** —

```
(defun |whatSpad2Cmd,fixpat| (x)
 (let (xp)
  (if (and (consp x) (progn (setq xp (qcar x)) t))
   (downcase xp)
   (downcase x))))
```

————-

[1] “display” (30.2.1 p 531) “set” (45.36.1 p 799) “show” (46.1.1 p 806)

53.1.4 defun whatSpad2Cmd

[reportWhatOptions p931]
[selectOptionLC p475]
[sayKeyedMsg p327]
[seq p??]
[exit p??]
[whatSpad2Cmd,fixpat p929]
[whatSpad2Cmd p930]
[filterAndFormatConstructors p934]
[whatCommands p931]
[apropos p935]
[printSynonyms p470]
[$e p??]
[$whatOptions p929]

— defun whatSpad2Cmd —

```
(defun |whatSpad2Cmd| (arg)
 (prog (|$e| |key0| key args)
  (declare (special |$e| |$whatOptions|))
  (return
   (seq
    (progn
     (setq |$e| |$EmptyEnvironment|)
     (cond
      ((null arg) (|reportWhatOptions|))
      (t
        (setq |key0| (car arg))
        (setq args (cdr arg))
        (setq key (|selectOptionLC| |key0| |$whatOptions| nil))
        (cond
         ((null key) (|sayKeyedMsg| 's2iz0043 nil))
         (t
          (setq args
           (prog (t0)
            (setq t0 nil)
            (return
              (do ((t1 args (cdr t1)) (p nil))
                  ((or (atom t1)
                       (progn (setq p (car t1)) nil))
                   (nreverse0 t0))
                (seq
                 (exit
                  (setq t0 (cons (|whatSpad2Cmd,fixpat| p) t0))))))))
          (seq
           (cond
            ((eq key '|things|)
```

```
  (do ((t2 |$whatOptions| (cdr t2)) (opt nil))
      ((or (atom t2) (progn (setq opt (CAR t2)) nil)) nil)
   (seq
    (exit
     (cond
      ((null (member opt '(|things|)))
        (exit (|whatSpad2Cmd| (cons opt args)))))))))
((eq key '|categories|)
  (|filterAndFormatConstructors| '|category| "Categories" args))
((eq key '|commands|) (|whatCommands| args))
((eq key '|domains|)
  (|filterAndFormatConstructors| '|domain| "Domains" args))
((eq key '|operations|)
   (|apropos| args))
((eq key '|packages|)
  (|filterAndFormatConstructors| '|package| "Packages" args))
(t
 (cond ((eq key '|synonyms|)
         (|printSynonyms| args)))))))))))))))
```

———-

53.1.5 defun Show keywords for)what command

[sayBrightly p**??**]
[$whatOptions p929]

— **defun reportWhatOptions** —

```
(defun |reportWhatOptions| ()
 (let (optlist)
 (declare (special |$whatOptions|))
  (setq optlist
   (reduce #'append
    (mapcar #'(lambda (x) `(|%l| "        " ,x)) |$whatOptions|)))
  (|sayBrightly|
   `(|%b| "  )what" |%d| "argument keywords are" |%b| ,@optlist |%d|
     |%l| "   or abbreviations thereof." |%l| |%l| "   Issue" |%b| ")what ?"
     |%d| "for more information."))))
```

———-

53.1.6 defun The)what commands implementation

[centerAndHighlight p**??**]
[strconc p**??**]

[specialChar p970]
[filterListOfStrings p932]
[commandsForUserLevel p444]
[sayMessage p??]
[blankList p??]
[sayAsManyPerLineAsPossible p??]
[say p??]
[sayKeyedMsg p327]
[$systemCommands p439]
[$linelength p765]
[$UserLevel p798]

— **defun whatCommands** —

```
(defun |whatCommands| (patterns)
 (let (label ell)
 (declare (special |$systemCommands| $linelength |$UserLevel|))
  (setq label
   (strconc '|System Commands for User Level: |
     (princ-to-string |$UserLevel|)))
  (|centerAndHighlight| label $linelength (|specialChar| '|hbar|))
  (setq ell
   (|filterListOfStrings| patterns
    (mapcar #'princ-to-string (|commandsForUserLevel| |$systemCommands|))))
  (when patterns
   (if ell
     (|sayMessage|
      `("System commands at this level matching patterns:" |%l| "   " |%b|
         ,@(append (|blankList| patterns) (list '|%d|))))
     (|sayMessage|
      `("No system commands at this level matching patterns:" |%l| "   " |%b|
        ,@(append (|blankList| patterns) (list '|%d|))))))
  (when ell
   (|sayAsManyPerLineAsPossible| ell)
   (say " "))
  (unless patterns (|sayKeyedMsg| 's2iz0046 nil))))
```

———-

53.1.7 defun Find all names contained in a pattern

Names and patterns are lists of strings. This returns a list of strings in names that contains any of the strings in the patterns [satisfiesRegularExpressions p933]

— **defun filterListOfStrings** —

```
(defun |filterListOfStrings| (patterns names)
```

```
 (let (result)
 (if (or (null patterns) (null names))
  names
  (dolist (name (reverse names) result)
   (when (|satisfiesRegularExpressions| name patterns)
    (push name result))))))
```

———-

53.1.8 defun Find function of names contained in pattern

The argument names and patterns are lists of strings. The argument fn is something like CAR or CADR This returns a list of strings in names that contains any of the strings in patterns [satisfiesRegularExpressions p933]

— **defun filterListOfStringsWithFn** —

```
(defun |filterListOfStringsWithFn| (patterns names fn)
 (let (result)
 (if (or (null patterns) (null names))
  names
  (dolist (name (reverse names) result)
   (when (|satisfiesRegularExpressions| (funcall fn name) patterns)
    (push name result))))))
```

———-

53.1.9 defun satisfiesRegularExpressions

[strpos p1039]

— **defun satisfiesRegularExpressions** —

```
(defun |satisfiesRegularExpressions| (name patterns)
 (let ((dname (downcase (copy name))))
  (dolist (pattern patterns)
   (when (strpos pattern dname 0 "@")
    (return-from nil t)))))
```

———-

53.1.10 defun filterAndFormatConstructors

[sayMessage p??]
[blankList p??]
[pp2Cols p??]
[centerAndHighlight p??]
[specialChar p970]
[filterListOfStringsWithFn p933]
[whatConstructors p935]
[function p??]
[$linelength p765]

— defun filterAndFormatConstructors —

```
(defun |filterAndFormatConstructors| (constrType label patterns)
 (prog (l)
 (declare (special $linelength))
  (return
   (progn (|centerAndHighlight| label $linelength (|specialChar| '|hbar|))
    (setq l
     (|filterListOfStringsWithFn| patterns
       (|whatConstructors| constrType)
       (|function| cdr)))
    (cond (patterns
     (cond
      ((null l)
       (|sayMessage|
        (cons "   No "
         (cons label
          (cons " with names matching patterns:"
           (cons '|%l|
            (cons "   "
             (cons '|%b|
              (append (|blankList| patterns)
               (cons '|%d| nil)))))))))
      (t
        (|sayMessage|
         (cons label
          (cons " with names matching patterns:"
           (cons '|%l|
            (cons "   "
             (cons '|%b|
              (append (|blankList| patterns)
               (cons '|%d| nil))))))))))))
    (cond (l (|pp2Cols| l)))))))
```

———-

53.1.11 defun whatConstructors

[boot-equal p??]
[getdatabase p1001]
[seq p??]
[msort p??]
[exit p??]

— **defun whatConstructors** —

```
(defun |whatConstructors| (constrType)
 (prog nil
  (return
   (seq
    (msort
     (prog (t0)
      (setq t0 nil)
      (return
       (do ((t1 (|allConstructors|) (cdr t1)) (|con| nil))
           ((or (atom t1) (progn (setq |con| (car t1)) nil)) (nreverse0 t0))
        (seq
         (exit
          (cond
           ((equal (getdatabase |con| 'constructorkind) constrType)
             (setq t0
              (cons
               (cons
                (getdatabase |con| 'abbreviation)
                (string |con|))
               t0)))))))))))))
```

———-

53.1.12 Display all operation names containing the fragment

Argument l is a list of operation name fragments. This displays all operation names containing these fragments [allOperations p1027]
[filterListOfStrings p932]
[seq p??]
[exit p??]
[downcase p??]
[sayMessage p??]
[sayAsManyPerLineAsPossible p??]
[msort p??]
[sayKeyedMsg p327]

— defun apropos —

```
(defun |apropos| (arg)
 "Display all operation names containing the fragment"
 (prog (ops)
  (return
   (seq
    (progn
     (setq ops
      (cond
       ((null arg) (|allOperations|))
       (t
         (|filterListOfStrings|
          (prog (t0)
           (setq t0 nil)
           (return
            (do ((t1 arg (cdr t1)) (p nil))
                ((or (atom t1) (progn (setq p (car t1)) nil))
                  (nreverse0 t0))
             (seq (exit (setq t0 (cons (downcase (princ-to-string p)) t0)))))))
          (|allOperations|)))))
     (cond
      (ops
       (|sayMessage| "Operations whose names satisfy the above pattern(s):")
       (|sayAsManyPerLineAsPossible| (msort ops))
       (|sayKeyedMsg| 's2if0011 (cons (car ops) nil)))
      (t
       (|sayMessage| "   There are no operations containing those patterns")
       nil)))))))
```

———-

Chapter 54

)with help page Command

54.1 with help page man page

— **with.help** —

```
This command is obsolete.
This has been renamed )library.

See also:
o )library
```

————-

[1]

54.1.1 defun with

[library p1005]

— **defun with** —

```
(defun |with| (args)
 (|library| args))
```

————-

[1] "library" (65.1.34 p 1005)

Chapter 55

)workfiles help page Command

55.1 workfiles help page man page

55.1.1 defun workfiles

[workfilesSpad2Cmd p939]

— **defun workfiles** —

```
(defun |workfiles| (l)
 (|workfilesSpad2Cmd| l))
```

———-

55.1.2 defun workfilesSpad2Cmd

[throwKeyedMsg p??]
[selectOptionLC p475]
[pathname p1036]
[delete p??]
[makeInputFilename p973]
[sayKeyedMsg p327]
[namestring p1034]
[updateSourceFiles p542]
[say p??]
[centerAndHighlight p??]
[specialChar p970]
[sortby p??]
[sayBrightly p??]

[$options p**??**]
[$sourceFiles p**??**]
[$linelength p765]

— defun workfilesSpad2Cmd —

```
(defun |workfilesSpad2Cmd| (args)
 (let (deleteflag type flist type1 fl)
  (declare (special |$options| |$sourceFiles| $linelength))
  (cond
   (args (|throwKeyedMsg| 's2iz0047 nil))
   (t
     (setq deleteflag nil)
     (do ((t0 |$options| (cdr t0)) (t1 nil))
         ((or (atom t0)
              (progn (setq t1 (car t0)) nil)
              (progn (progn (setq type (car t1)) t1) nil))
          nil)
       (setq type1
          (|selectOptionLC| type '(|boot| |lisp| |meta| |delete|) nil))
       (cond
        ((null type1) (|throwKeyedMsg| 's2iz0048 (cons type nil)))
        ((eq type1 '|delete|) (setq deleteflag t))))
     (do ((t2 |$options| (cdr t2)) (t3 nil))
         ((or (atom t2)
          (progn (setq t3 (CAR t2)) nil)
          (progn
           (progn
            (setq type (car t3))
            (setq flist (cdr t3)) t3)
           nil))
          nil)
      (setq type1 (|selectOptionLC| type '(|boot| |lisp| |meta| |delete|) nil))
      (unless (eq type1 '|delete|)
        (dolist (file flist)
         (setq fl (|pathname| (list file type1 "*")))
         (cond
          (deleteflag
           (setq |$sourceFiles| (|delete| fl |$sourceFiles|)))
          ((null (makeInputFilename fl))
           (|sayKeyedMsg| 's2iz0035 (list (|namestring| fl))))
          (t (|updateSourceFiles| fl))))))
     (say " ")
     (|centerAndHighlight|
       '| User-specified work files |
       $linelength
       (|specialChar| '|hbar|))
     (say " ")
     (if (null |$sourceFiles|)
```

```
(say "   no files specified")
(progn
  (setq |$sourceFiles| (sortby '|pathnameType| |$sourceFiles|))
  (do ((t5 |$sourceFiles| (cdr t5)) (fl nil))
      ((or (atom t5) (progn (setq fl (car t5)) nil)) nil)
     (|sayBrightly| (list "   " (|namestring| fl)))))))))))
```

———-

Chapter 56

)zsystemdevelopment help page Command

56.1 zsystemdevelopment help page man page

56.1.1 defun zsystemdevelopment

[zsystemDevelopmentSpad2Cmd p943]

— **defun zsystemdevelopment** —

```
(defun |zsystemdevelopment| (arg)
 (|zsystemDevelopmentSpad2Cmd| arg))
```

———-

56.1.2 defun zsystemDevelopmentSpad2Cmd

[zsystemdevelopment1 p944]
[$InteractiveMode p24]

— **defun zsystemDevelopmentSpad2Cmd** —

```
(defun |zsystemDevelopmentSpad2Cmd| (arg)
 (declare (special |$InteractiveMode|))
 (|zsystemdevelopment1| arg |$InteractiveMode|))
```

———-

56.1.3 defun zsystemdevelopment1

[filenam p??]
[selectOptionLC p475]
[/D,1 p??]
[/comp p??]
[version p??]
[defiostream p972]
[next p38]
[shut p972]
[kar p??]
[kadr p??]
[kaddr p??]
[sayMessage p??]
[sayBrightly p??]
[bright p??]
[$InteractiveMode p24]
[$options p??]
[/wsname p??]
[/version p??]

— **defun zsystemdevelopment1** —

```
(defun |zsystemdevelopment1| (arg im)
 (let (|$InteractiveMode| fromopt opt optargs newopt opt1 constream upf fun)
 (declare (special |$InteractiveMode| /wsname /version |$options|))
 (setq |$InteractiveMode| im)
 (setq fromopt nil)
 (do ((t0 |$options| (cdr t0)) (t1 nil))
     ((or (atom t0)
          (progn (setq t1 (car t0)) nil)
          (progn
           (progn
            (setq opt (CAR t1))
            (setq optargs (CDR t1))
            t1)
           nil))
       nil)
   (setq opt1 (|selectOptionLC| opt '(|from|) nil))
   (when (eq opt1 '|from|) (setq fromopt (cons (cons 'from optargs) nil))))
 (do ((t2 |$options| (cdr t2)) (t3 nil))
     ((or (atom t2)
          (progn (setq t3 (car t2)) nil)
          (progn
           (progn
            (setq opt (car t3))
            (setq optargs (cdr t3))
            t3)
```

```
          nil))
     nil)
 (unless optargs (setq optargs arg))
 (setq newopt (append optargs fromopt))
 (setq opt1 (|selectOptionLC| opt '(|from|) nil))
 (cond
  ((eq opt1 '|from|)   nil)
  ((eq opt '|c|)       (|/D,1| newopt (/COMP) nil nil))
  ((eq opt '|d|)       (|/D,1| newopt 'define nil nil))
  ((eq opt '|dt|)      (|/D,1| newopt 'define nil t))
  ((eq opt '|ct|)      (|/D,1| newopt (/COMP) nil t))
  ((eq opt '|ctl|)     (|/D,1| newopt (/COMP) nil 'tracelet))
  ((eq opt '|ec|)      (|/D,1| newopt (/COMP) t nil))
  ((eq opt '|ect|)     (|/D,1| newopt (/COMP) t t))
  ((eq opt '|e|)       (|/D,1| newopt nil t nil))
  ((eq opt '|version|) (|version|))
  ((eq opt '|pause|)
    (setq constream
      (defiostream '((device . console) (qual . v)) 120 0))
    (next constream)
    (shut constream))
  ((or
      (eq opt '|update|)
      (eq opt '|patch|))
    (setq |$InteractiveMode| nil)
    (setq upf
     (cons
      (or (kar optargs) /version)
      (cons
       (or (kadr optargs) /wsname)
       (cons (or (kaddr optargs) '*) nil))))
    (setq fun
     (cond
      ((eq opt '|patch|) '/update-lib-1)
      (t '/update-1)))
    (catch 'filenam (funcall fun upf))
    (|sayMessage| "   Update/patch is completed."))
  ((null optargs)
   (|sayBrightly| `("   An argument is required for" ,@(|bright| opt))))
  (t
   (|sayMessage|
    `("   Unknown option:" ,@(|bright| opt)
      |%l| "   Available options are"
      ,@(|bright|
         "c ct e ec ect cls pause update patch compare record"))))))))
```

———-

Chapter 57

Handlers for Special Forms

This file contains the functions which do type analysis and evaluation of special functions in the interpreter. Special functions are ones which are not defined in the algebra code, such as assignment, construct, COLLECT and declaration.

Operators which require special handlers all have a LISP "up" property which is the name of the special handler, which is always the word "up" followed by the operator name. If an operator has this "up" property the handler is called automatically from bottomUp instead of general modemap selection.

The up handlers are usually split into two pieces, the first is the up function itself, which performs the type analysis, and an "eval" function, which generates (and executes, if required) the code for the function.

The up functions always take a single argument, which is the entire attributed tree for the operation, and return the modeSet of the node, which is a singleton list containing the type computed for the node.

The eval functions can take any arguments deemed necessary. Actual evaluation is done if `$genValue` is true, otherwise code is generated.

(See the function analyzeMap for other things that may affect what is generated in these functions.)

These functions are required to do two things:

1. do a putValue on the operator vector with the computed value of the node, which is a triple. This is usually done in the eval functions.
2. do a putModeSet on the operator vector with a list of the computed type of the node. This is usually done in the up functions.

There are several special modes used in these functions:

1. Void is the mode that should be used for all statements that do not otherwise return values, such as declarations, loops, IF-THEN's without ELSE's, etc..

2. `$NoValueMode` and `$ThrowAwayMode` used to be used in situations where Void is now used, and are being phased out completely.

57.0.4 defun getAndEvalConstructorArgument

[getValue p??]
[objMode p??]
[isWrapped p??]
[objVal p??]
[isLocalVar p??]
[compFailure p??]
[objNewWrap p??]
[timedEVALFUN p??]

— **defun getAndEvalConstructorArgument** —

```
(defun |getAndEvalConstructorArgument| (tree)
 (let (triple)
  (setq triple (|getValue| tree))
  (cond
   ((eq (|objMode| triple) '(|Domain|)) triple)
   ((|isWrapped| (|objVal| triple)) triple)
   ((|isLocalVar| (|objVal| triple))
     (|compFailure| "   Local variable or parameter used in type"))
   (t
    (|objNewWrap| (|timedEVALFUN| (|objVal| triple)) (|objMode| triple))))))
```

———-

57.0.5 defun replaceSharps

Replaces all sharps in x by the arguments of domain d. Replaces all replaces the triangle variables [subCopy p??]
[$TriangleVariableList p??]
[$FormalMapVariableList p??]

— **defun replaceSharps** —

```
(defun |replaceSharps| (x d)
 (let (sl)
 (declare (special |$TriangleVariableList| |$FormalMapVariableList|))
  (loop for e in (rest d) for var in |$FormalMapVariableList|
   do (setq sl (cons (cons var e) sl)))
  (setq x (|subCopy| x sl))
  (setq sl nil)
```

```
  (loop for e in (rest d) for var in |$TriangleVariableList|
   do (setq sl (cons (cons var e) sl)))
  (|subCopy| x sl)))
```

———-

57.0.6 defun isDomainValuedVariable

Returns the value of form if form is a variable with a type value [identp p1040]
[get p??]
[member p1042]
[objMode p??]
[objValUnwrap p??]
[$e p??]
[$env p??]
[$InteractiveFrame p??]

— **defun isDomainValuedVariable** —

```
(defun |isDomainValuedVariable| (form)
 (let (val)
 (declare (special |$e| |$env| |$InteractiveFrame|))
  (when (and (identp form)
             (setq val
               (or (|get| form '|value| |$InteractiveFrame|)
                   (and (consp |$env|)  (|get| form '|value| |$env|))
                   (and (consp |$e|)    (|get| form '|value| |$e|))))
             (|member| (|objMode| val) '((|Domain|) (|SubDomain| (|Domain|)))))
    (|objValUnwrap| val))))
```

———-

57.0.7 defun evalCategory

[ofCategory p417]
[isPartialMode p418]

— **defun evalCategory** —

```
(defun |evalCategory| (d c)
  (or (|isPartialMode| d) (|ofCategory| d c)))
```

———-

Chapter 58

Handling input files

58.0.8 defun Handle .axiom.input file

[/editfile p511]

— **defun readSpadProfileIfThere** —

```
(defun |readSpadProfileIfThere| ()
 (let ((file (list '|.axiom| '|input|)))
 (declare (special /editfile))
  (when (makeInputFilename file) (setq /editfile file) (/rq))))
```

———-

58.0.9 defvar $boot-line-stack

— **initvars** —

```
(defvar boot-line-stack nil "List of lines returned from preparse")
```

———-

58.0.10 defvar $in-stream

— **initvars** —

```
(defvar in-stream t "Current input stream.")
```

———-

58.0.11 defvar $out-stream

— **initvars** —

```
(defvar out-stream t "Current output stream.")
```

———-

58.0.12 defvar $file-closed

— **initvars** —

```
(defvar file-closed nil  "Way to stop EOF tests for console input.")
```

———-

58.0.13 defvar $echo-meta

— **initvars** —

```
(defvar echo-meta nil "T if you want a listing of what has been read.")
```

———-

58.0.14 defvar $noSubsumption

— **initvars** —

```
(defvar |$noSubsumption| t)
```

———-

58.0.15 defvar $envHashTable

The `$envHashTable` variable is a hashtable that optimizes lookups in the environment, which normally involve search. This gets populated in the addBinding function.

— **initvars** —

```
(defvar |$envHashTable| nil)
```

————-

58.0.16 defun Dynamically add bindings to the environment

[getProplist p954]
[addBindingInteractive p957]
[hput p1038]
[$InteractiveMode p24]
[$envHashTable p953]

— **defun addBinding** —

```
(defun |addBinding| (var proplist e)
 (let (tailContour tailEnv tmp1 curContour lx)
 (declare (special |$InteractiveMode| |$envHashTable|))
  (setq curContour (caar e))
  (setq tailContour (cdar e))
  (setq tailEnv (cdr e))
  (cond
   ((eq proplist (|getProplist| var e)) e)
   (t
    (when |$envHashTable|
      (do ((prop proplist (cdr prop)) (u nil))
          ((or (atom prop)
               (progn (setq u (car prop)) nil))
           nil)
        (hput |$envHashTable| (list var (car u)) t)))
    (cond
     (|$InteractiveMode| (|addBindingInteractive| var proplist e))
     (t
      (when (and (consp curContour)
                 (progn
                  (setq tmp1 (qcar curContour))
                  (and (consp tmp1) (equal (qcar tmp1) var))))
        (setq curContour (cdr curContour)))
      (setq lx (cons var proplist))
      (cons (cons (cons lx curContour) tailContour) tailEnv)))))))
```

————-

58.0.17 defun Fetch a property list for a symbol from CategoryFrame

[getProplist p954]
[search p954]
[$CategoryFrame p??]

— **defun getProplist** —

```
(defun |getProplist| (x e)
 (let (u pl)
 (declare (special |$CategoryFrame|))
  (cond
   ((null (atom x)) (|getProplist| (car x) e))
   ((setq u (|search| x e)) u)
   ((setq pl (|search| x |$CategoryFrame|)) pl))))
```

———-

58.0.18 defun Search for a binding in the environment list

[searchCurrentEnv p954]
[searchTailEnv p955]

— **defun search** —

```
(defun |search| (x e)
 (let ((curEnv (car e)) (tailEnv (cdr e)))
  (or (|searchCurrentEnv| x curEnv) (|searchTailEnv| x tailEnv))))
```

———-

58.0.19 defun Search for a binding in the current environment

```
searchCurrentEnv(x,currentEnv) ==
  for contour in currentEnv repeat
    if u:= ASSQ(x,contour) then return (signal:= u)
  KDR signal
```

[assq p1044]
[kdr p??]

— **defun searchCurrentEnv** —

```
(defun |searchCurrentEnv| (x currentEnv)
```

```
(prog (u signal)
 (return
  (seq
   (progn
    (do ((thisenv currentEnv (cdr thisenv)) (contour nil))
        ((or (atom thisenv) (progn (setq contour (car thisenv)) nil)) nil)
     (seq
      (exit
       (cond
        ((setq u (assq x contour)) (return (setq signal u)))
        (t nil)))))
    (kdr signal))))))
```

———-

58.0.20 defun searchTailEnv

```
;searchTailEnv(x,e) ==
;  for env in e repeat
;    signal:=
;      for contour in env repeat
;        if (u:= ASSQ(x,contour)) and ASSQ("FLUID",u) then return (signal:= u)
;      if signal then return signal
;  KDR signal
```

[assq p1044]
[kdr p??]

— **defun searchTailEnv** —

```
(defun |searchTailEnv| (x e)
 (prog (u signal)
  (return
   (seq
    (progn
     (do ((thise e (cdr thise)) (env nil))
         ((or (atom thise) (progn (setq env (car thise)) nil)) nil)
       (seq
        (exit
         (setq signal
          (progn
           (do ((cone env (cdr cone)) (contour nil))
               ((or (atom cone) (progn (setq contour (car cone)) nil)) nil)
             (seq
              (exit
               (cond
                ((and (setq u (assq x contour)) (assq 'fluid u))
```

```
                (return (setq signal u)))
               (t nil)))))
         (cond
          (signal (return signal))
          (t nil)))))))
   (kdr signal))))))
```

Chapter 59

File Parsing

59.0.21 defun Bind a variable in the interactive environment

[assq p1044]

— **defun addBindingInteractive** —

```
(defun |addBindingInteractive| (var proplist e)
 (let ((curContour (caar e)) u)
  (cond
   ((setq u (assq var curContour)) (rplacd u proplist) e)
   (t (rplac (caar e) (cons (cons var proplist) curContour)) e))))
```

————-

59.0.22 defvar $line-handler

— **initvars** —

```
(defparameter line-handler 'next-META-line "Who grabs lines for us.")
```

————-

59.0.23 defvar $spad-errors

— **initvars** —

```
(defvar $spad_errors (vector 0 0 0))
```

———-

59.0.24 defvar $xtokenreader

— **initvars** —

```
(defvar xtokenreader 'spadtok)
```

———-

59.0.25 defun Initialize the spad reader

[next-lines-clear p962]
[ioclear p962]
[$spad-errors p957]
[spaderrorstream p**??**]
[*standard-output* p**??**]
[xtokenreader p958]
[line-handler p957]
[meta-error-handler p**??**]
[file-closed p952]
[boot-line-stack p951]

— **defun init-boot/spad-reader** —

```
(defun init-boot/spad-reader ()
 (declare (special $spad_errors spaderrorstream *standard-output*
           xtokenreader line-handler meta-error-handler file-closed
           boot-line-stack))
  (setq $spad_errors (vector 0 0 0))
  (setq spaderrorstream *standard-output*)
  (setq xtokenreader 'get-BOOT-token)
  (setq line-Handler 'next-BOOT-line)
  (setq meta-error-handler 'spad-syntax-error)
  (setq file-closed nil)
  (next-lines-clear)
  (ioclear))
```

———-

59.0.26 defun spad-syntax-error

[bumperrorcount p??]
[consoleinputp p??]
[spad-long-error p959]
[spad-short-error p960]
[ioclear p962]
[debugmode p??]
[spad-reader p??]

— **defun spad-syntax-error** —

```
(defun spad-syntax-error (&rest byebye)
  "Print syntax error indication, underline character, scrub line."
  (declare (special debugmode))
  (bumperrorcount '|syntax|)
  (cond ((and (eq debugmode 'yes) (not(consoleinputp in-stream)))
         (spad-long-error))
        ((spad-short-error)))
  (ioclear)
  (throw 'spad_reader nil))
```

———-

59.0.27 defun spad-long-error

[spad-error-loc p960]
[iostat p960]
[out-stream p952]
[spaderrorstream p??]

— **defun spad-long-error** —

```
(defun spad-long-error ()
 (declare (special spaderrorstream))
  (spad-error-loc spaderrorstream)
  (iostat)
  (unless (equal out-stream spaderrorstream)
    (spad-error-loc out-stream)
    (terpri out-stream)))
```

———-

59.0.28 defun spad-short-error

[line-past-end-p p??]
[line-print p??]
[$current-line p??]

— **defun spad-short-error** —

```
(defun spad-short-error ()
 (if (line-past-end-p Current-Line)
    (format t "~&The current line is empty.~%")
    (progn
     (format t "~&The current line is:~%~%")
     (line-print current-line))))
```

———-

59.0.29 defun spad-error-loc

— **defun spad-error-loc** —

```
(defun spad-error-loc (str)
  (format str "******** Boot Syntax Error detected ********"))
```

———-

59.0.30 defun iostat

[line-past-end-p p??]
[line-print p??]
[token-stack-show p961]
[next-lines-show p961]
[$boot p25]
[$spad p20]
[$current-line p??]

— **defun iostat** —

```
(defun iostat ()
  "Tell me what the current state of the parsing world is."
  (declare (special $boot $spad))
  (if (line-past-end-p Current-Line)
    (format t "~&The current line is empty.~%")
```

```
    (progn
     (format t "~&The current line is:~%~%")
     (line-print current-line)))
  (if (or $boot $spad) (next-lines-show))
  (token-stack-show)
  nil)
```

———-

59.0.31 defun next-lines-show

[boot-line-stack p951]

— **defun next-lines-show** —

```
(defun next-lines-show ()
  (declare (special boot-line-stack))
  (and boot-line-stack (format t "Currently preparsed lines are:~%~%"))
  (mapcar #'(lambda (line)
              (format t "~&~5D> ~A~%" (car line) (cdr Line)))
          boot-line-stack))
```

———-

59.0.32 defun token-stack-show

[token-type p**??**]
[valid-tokens p**??**]
[current-token p**??**]
[next-token p**??**]
[prior-token p**??**]

— **defun token-stack-show** —

```
(defun token-stack-show ()
  (if (= valid-tokens 0)
   (format t "~%There are no valid tokens.~%")
   (format t "~%The number of valid tokens is ~S.~%" valid-tokens))
  (when (> valid-tokens 0)
   (format t "The current token is~%")
   (describe current-token))
  (when (> valid-tokens 1)
   (format t "The next token is~%")
   (describe next-token))
  (when (token-type prior-token)
```

```
  (format t "The prior token was~%")
  (describe prior-token)))
```

———-

59.0.33 defun ioclear

The IO state manipulation routines assume that

- one I/O stream pair is in effect at any moment
- there is a current line
- there is a current token and a next token
- there is a reduction stack

[line-clear p**??**]
[reduce-stack-clear p**??**]
[current-fragment p**??**]
[current-line p**??**]
[$boot p25]
[$spad p20]

— **defun ioclear** —

```
(defun ioclear (&optional (in t) (out t))
 (declare (special current-fragment current-line $boot $spad)
          (ignore in out))
  (setq current-fragment nil)
  (line-clear current-line)
  (token-install nil nil current-token nil)
  (token-install nil nil next-token nil)
  (token-install nil nil prior-token nil)
  (reduce-stack-clear)
  (if (or $boot $spad) (next-lines-clear))
  nil)
```

———-

59.0.34 defun Set boot-line-stack to nil

[boot-line-stack p951]

— **defun next-lines-clear** —

```
(defun next-lines-clear ()
 (setq boot-line-stack nil))
```

———————-

Chapter 60

Handling output

60.1 Special Character Tables

60.1.1 defvar $defaultSpecialCharacters

— **initvars** —

```
(defvar |$defaultSpecialCharacters| (list
 (int-char 28)    ; upper left corner
 (int-char 27)    ; upper right corner
 (int-char 30)    ; lower left corner
 (int-char 31)    ; lower right corner
 (int-char 79)    ; vertical bar
 (int-char 45)    ; horizontal bar
 (int-char 144)   ; APL quad
 (int-char 173)   ; left bracket
 (int-char 189)   ; right bracket
 (int-char 192)   ; left brace
 (int-char 208)   ; right brace
 (int-char 59)    ; top    box tee
 (int-char 62)    ; bottom box tee
 (int-char 63)    ; right  box tee
 (int-char 61)    ; left   box tee
 (int-char 44)    ; center box tee
 (int-char 224))) ; back slash
```

———-

60.1.2 defvar $plainSpecialCharacters0

— **initvars** —

```
(defvar |$plainSpecialCharacters0| (list
 (int-char 78)     ; upper left corner   (+)
 (int-char 78)     ; upper right corner  (+)
 (int-char 78)     ; lower left corner   (+)
 (int-char 78)     ; lower right corner  (+)
 (int-char 79)     ; vertical bar
 (int-char 96)     ; horizontal bar      (-)
 (int-char 111)    ; APL quad            (?)
 (int-char 173)    ; left bracket
 (int-char 189)    ; right bracket
 (int-char 192)    ; left brace
 (int-char 208)    ; right brace
 (int-char 78)     ; top    box tee      (+)
 (int-char 78)     ; bottom box tee      (+)
 (int-char 78)     ; right  box tee      (+)
 (int-char 78)     ; left   box tee      (+)
 (int-char 78)     ; center box tee      (+)
 (int-char 224)))  ; back slash
```

———-

60.1.3 defvar $plainSpecialCharacters1

— **initvars** —

```
(defvar |$plainSpecialCharacters1| (list
 (int-char 107)    ; upper left corner   (,)
 (int-char 107)    ; upper right corner  (,)
 (int-char 125)    ; lower left corner   (')
 (int-char 125)    ; lower right corner  (')
 (int-char 79)     ; vertical bar
 (int-char 96)     ; horizontal bar      (-)
 (int-char 111)    ; APL quad            (?)
 (int-char 173)    ; left bracket
 (int-char 189)    ; right bracket
 (int-char 192)    ; left brace
 (int-char 208)    ; right brace
 (int-char 78)     ; top    box tee      (+)
 (int-char 78)     ; bottom box tee      (+)
 (int-char 78)     ; right  box tee      (+)
 (int-char 78)     ; left   box tee      (+)
```

```
 (int-char 78)     ; center box tee      (+)
 (int-char 224)))  ; back slash
```

———-

60.1.4 defvar $plainSpecialCharacters2

— **initvars** —

```
(defvar |$plainSpecialCharacters2| (list
 (int-char 79)     ; upper left corner   (|)
 (int-char 79)     ; upper right corner  (|)
 (int-char 79)     ; lower left corner   (|)
 (int-char 79)     ; lower right corner  (|)
 (int-char 79)     ; vertical bar
 (int-char 96)     ; horizontal bar      (-)
 (int-char 111)    ; APL quad            (?)
 (int-char 173)    ; left bracket
 (int-char 189)    ; right bracket
 (int-char 192)    ; left brace
 (int-char 208)    ; right brace
 (int-char 78)     ; top    box tee      (+)
 (int-char 78)     ; bottom box tee      (+)
 (int-char 78)     ; right  box tee      (+)
 (int-char 78)     ; left   box tee      (+)
 (int-char 78)     ; center box tee      (+)
 (int-char 224)))  ; back slash
```

———-

60.1.5 defvar $plainSpecialCharacters3

— **initvars** —

```
(defvar |$plainSpecialCharacters3| (list
 (int-char 96)     ; upper left corner   (-)
 (int-char 96)     ; upper right corner  (-)
 (int-char 96)     ; lower left corner   (-)
 (int-char 96)     ; lower right corner  (-)
 (int-char 79)     ; vertical bar
 (int-char 96)     ; horizontal bar      (-)
 (int-char 111)    ; APL quad            (?)
 (int-char 173)    ; left bracket
```

```
 (int-char 189)    ; right bracket
 (int-char 192)    ; left brace
 (int-char 208)    ; right brace
 (int-char 78)     ; top    box tee      (+)
 (int-char 78)     ; bottom box tee      (+)
 (int-char 78)     ; right  box tee      (+)
 (int-char 78)     ; left   box tee      (+)
 (int-char 78)     ; center box tee      (+)
 (int-char 224)))  ; back slash
```

———-

60.1.6 defvar $plainRTspecialCharacters

— **initvars** —

```
(defvar |$plainRTspecialCharacters| (list
 (QUOTE +)       ; upper left corner   (+)
 (QUOTE +)       ; upper right corner  (+)
 (QUOTE +)       ; lower left corner   (+)
 (QUOTE +)       ; lower right corner  (+)
 (QUOTE |\||)    ; vertical bar
 (QUOTE -)       ; horizontal bar      (-)
 (QUOTE ?)       ; APL quad            (?)
 (QUOTE [)       ; left bracket
 (QUOTE ])       ; right bracket
 (QUOTE {)       ; left brace
 (QUOTE })       ; right brace
 (QUOTE +)       ; top    box tee      (+)
 (QUOTE +)       ; bottom box tee      (+)
 (QUOTE +)       ; right  box tee      (+)
 (QUOTE +)       ; left   box tee      (+)
 (QUOTE +)       ; center box tee      (+)
 (QUOTE |\\|)))  ; back slash
```

———-

60.1.7 defvar $RTspecialCharacters

— **initvars** —

```
(defvar |$RTspecialCharacters| (list
 (intern (string (code-char 218))) ;-- upper left corner   (+)
```

```
 (intern (string (code-char 191))) ;-- upper right corner  (+)
 (intern (string (code-char 192))) ;-- lower left corner   (+)
 (intern (string (code-char 217))) ;-- lower right corner  (+)
 (intern (string (code-char 179))) ;-- vertical bar
 (intern (string (code-char 196))) ;-- horizontal bar      (-)
 (list (code-char #x1d) (code-char #xe2))
                                   ;-- APL quad           (?)
 (QUOTE [)                         ;-- left bracket
 (QUOTE ])                         ;-- right bracket
 (QUOTE {)                         ;-- left brace
 (QUOTE })                         ;-- right brace
 (intern (string (code-char 194))) ;-- top    box tee      (+)
 (intern (string (code-char 193))) ;-- bottom box tee      (+)
 (intern (string (code-char 180))) ;-- right  box tee      (+)
 (intern (string (code-char 195))) ;-- left   box tee      (+)
 (intern (string (code-char 197))) ;-- center box tee      (+)
 (QUOTE |\\|)))                    ;-- back slash
```

———-

60.1.8 defvar $specialCharacters

— **initvars** —

```
(defvar |$specialCharacters| |$RTspecialCharacters|)
```

———-

60.1.9 defvar $specialCharacterAlist

— **initvars** —

```
(defvar |$specialCharacterAlist|
 '((|ulc| . 0)
   (|urc| . 1)
   (|llc| . 2)
   (|lrc| . 3)
   (|vbar| . 4)
   (|hbar| . 5)
   (|quad| . 6)
   (|lbrk| . 7)
   (|rbrk| . 8)
   (|lbrc| . 9)
```

```
   (|rbrc| . 10)
   (|ttee| . 11)
   (|btee| . 12)
   (|rtee| . 13)
   (|ltee| . 14)
   (|ctee| . 15)
   (|bslash| . 16)))
```

———-

60.1.10 defun Look up a special character code for a symbol

This function looks up a symbol in `$specialCharacterAlist`, gets the index into the EBCDIC table, and returns the appropriate character. **TPDHERE: Make this more international, not EBCDIC** [ifcdr p**??**]
[assq p1044]
[$specialCharacters p969]
[$specialCharacterAlist p969]

— **defun specialChar** —

```
(defun |specialChar| (symbol)
 (let (code)
 (declare (special |$specialCharacters| |$specialCharacterAlist|))
  (if (setq code (ifcdr (assq symbol |$specialCharacterAlist|)))
   (elt |$specialCharacters| code)
   "?")))
```

———-

Chapter 61

Stream and File Handling

61.0.11 defun make-instream

[makeInputFilename p973]

— **defun make-instream** —

```
(defun make-instream (filespec &optional (recnum 0))
 (declare (ignore recnum))
   (cond ((numberp filespec) (make-synonym-stream '*terminal-io*))
         ((null filespec) (error "not handled yet"))
         (t (open (makeInputFilename filespec)
                  :direction :input :if-does-not-exist nil))))
```

———-

61.0.12 defun make-outstream

[make-filename p**??**]

— **defun make-outstream** —

```
(defun make-outstream (filespec &optional (width nil) (recnum 0))
 (declare (ignore width) (ignore recnum))
   (cond ((numberp filespec) (make-synonym-stream '*terminal-io*))
         ((null filespec) (error "not handled yet"))
         (t (open (make-filename filespec) :direction :output))))
```

———-

61.0.13 defun make-appendstream

[make-filename p??]

— **defun make-appendstream** —

```
(defun make-appendstream (filespec &optional (width nil) (recnum 0))
 "fortran support"
 (declare (ignore width) (ignore recnum))
 (cond
  ((numberp filespec) (make-synonym-stream '*terminal-io*))
  ((null filespec) (error "make-appendstream: not handled yet"))
  ('else (open (make-filename filespec) :direction :output
          :if-exists :append :if-does-not-exist :create))))
```

———-

61.0.14 defun defiostream

— **defun defiostream** —

```
(defun defiostream (stream-alist buffer-size char-position)
 (declare (ignore buffer-size))
   (let ((mode (or (cdr (assoc 'mode stream-alist)) 'input))
         (filename (cdr (assoc 'file stream-alist)))
         (dev (cdr (assoc 'device stream-alist))))
      (if (eq dev 'console) (make-synonym-stream '*terminal-io*)
       (let ((strm (case mode
                         ((output o) (open (make-filename filename)
                                           :direction :output))
                         ((input i) (open (makeInputFilename filename)
                                           :direction :input)))))
         (if (and (numberp char-position) (> char-position 0))
           (file-position strm char-position))
           strm))))
```

———-

61.0.15 defun shut

[shut is-console (vol9)]

— **defun shut** —

```
(defun shut (st)
  (if (is-console st)
   st
   (if (streamp st) (close st) -1)))
```

———-

61.0.16 defun eofp

— **defun eofp** —

```
(defun eofp (stream) (null (peek-char nil stream nil nil)))
```

———-

61.0.17 defun makeStream

[make-appendstream p972]
[make-outstream p971]

— **defun makeStream** —

```
(defun |makeStream| (append filename i j)
 (if append
  (make-appendstream filename i j)
  (make-outstream filename i j)))
```

———-

61.0.18 defun Construct a new input file name

— **defun makeInputFilename** —

```
(defun makeInputFilename (filearg &optional (filetype nil))
 (let*
  ((filename  (make-filename filearg filetype))
   (dirname (pathname-directory filename))
   (ft (pathname-type filename))
   (dirs (getDirectoryList ft))
   (newfn nil))
  (if (or (null dirname) (eqcar dirname :relative))
```

```
   (dolist (dir dirs (probeName filename))
    (when (probe-file (setq newfn (concatenate 'string dir filename)))
     (return newfn)))
   (probeName filename))))
```

———-

61.0.19 defun getDirectoryList

[$current-directory p7]
[$UserLevel p798]
[$library-directory-list p9]
[$directory-list p8]

— **defun getDirectoryList** —

```
(defun getDirectoryList (ft &aux (cd (namestring $current-directory)))
 (declare (special $current-directory |$UserLevel| $library-directory-list
                   $directory-list))
 (if (member ft '("nrlib" "daase" "exposed") :test #'string=)
   (if (eq |$UserLevel| '|development|)
    (cons cd $library-directory-list)
    $library-directory-list)
   (adjoin cd
    (adjoin (namestring (user-homedir-pathname)) $directory-list
      :test #'string=)
    :test #'string=)))
```

———-

61.0.20 defun probeName

Sometimes we are given a file and sometimes we are given the name of an Axiom KAF (Keyed-Access File). KAF files are actually directories with a single file called “index.kaf”. We check for the latter case and return the directory name as the filename, per Axiom convention.

— **defun probeName** —

```
(defun probeName (file)
 (when (or (probe-file file)
           (probe-file (concatenate 'string (namestring file) "/index.kaf")))
 (namestring file)))
```

———-

61.0.21 defun makeFullNamestring

— **defun makeFullNamestring** —

```
(defun makeFullNamestring (filearg &optional (filetype nil))
  (namestring (merge-pathnames (make-filename filearg filetype))))
```

————-

61.0.22 defun Replace a file by erase and rename

[makeFullNamestring p975]

— **defun replaceFile** —

```
(defun replaceFile (filespec1 filespec2)
 ($erase (setq filespec1 (makeFullNamestring filespec1)))
 (rename-file (makeFullNamestring filespec2) filespec1))
```

————-

Chapter 62

The Spad Server Mechanism

62.0.23 defun openserver

This is a cover function for the C code used for communication interface.
— **defun openserver** —

```
(defun openserver (name)
  (open_server name))
```

———-

Chapter 63

Axiom Build-time Functions

63.0.24 defun spad-save

The **spad-save** function is just a cover function for more lisp system specific save functions. There is no standard name for saving a lisp image so we make one and conditionalize it at compile time.

This function is passed the name of an image that will be saved. The saved image contains all of the loaded functions.

This is used in the src/interp/Makefile.pamphlet in three places:

- creating depsys, an image for compiling axiom.

  Some of the Common Lisp code we compile uses macros which are assumed to be available at compile time. The **DEPSYS** image is created to contain the compile time environment and saved. We pipe compile commands into this environment to compile from Common Lisp to machine dependent code.

  ```
  DEPSYS=${OBJ}/${SYS}/bin/depsys
  ```

- creating savesys, an image for running axiom.

  Once we've compile all of the Common Lisp files we fire up a clean lisp image called **LOADSYS**, load all of the final executable code and save it out as **SAVESYS**. The **SAVESYS** image is copied to the `${MNT}/${SYS}/bin` subdirectory and becomes the axiom executable image.

  ```
  LOADSYS= ${OBJ}/${SYS}/bin/lisp
  SAVESYS= ${OBJ}/${SYS}/bin/interpsys
  AXIOMSYS= ${MNT}/${SYS}/bin/AXIOMsys
  ```

- creating debugsys, an image with all interpreted functions loaded.

Occasionally we need to really get into the system internals. The best way to do this is to run almost all of the lisp code interpreted rather than compiled (note that cfuns.lisp and sockio.lisp still need to be loaded in compiled form as they depend on the loader to link with lisp internals). This image is nothing more than a load of the file src/interp/debugsys.lisp.pamphlet. If you need to make test modifications you can add code to that file and it will show up here.

```
DEBUGSYS=${OBJ}/${SYS}/bin/debugsys
```

[save-system p**??**]
[$SpadServer p12]
[$openServerIfTrue p10]

— **defun spad-save** —

```
(defun user::spad-save (save-file)
  (declare (special |$SpadServer| $openServerIfTrue))
  (setq |$SpadServer| nil)
  (setq $openServerIfTrue t)
#+:AKCL
  (system::save-system save-file)
#+:allegro
  (if (fboundp 'boot::restart)
   (excl::dumplisp :name save-file :restart-function #'boot::restart)
   (excl::dumplisp :name save-file))
#+Lucid
  (if (fboundp 'boot::restart)
   (sys::disksave save-file :restart-function #'boot::restart)
   (sys::disksave save-file))
#+:CCL
  (preserve)
)
```

———-

Chapter 64

Exposure Groups

Exposure groups are a way of controlling the namespace available to the user. Certain algebra files are only useful for internal purposes but they contain functions have common names (like "map". In order to separate the user visible functions from the internal functions the algebra files are collected into "exposure groups". These large groups are grouped into sets in the variable `$globalExposureGroupAlist`.

Exposure group information is kept in the local frame. For more information "The Frame Mechanism" 33.3.1 on page 548.

Chapter 65

Databases

65.1 Database structure

In order to understand this program you need to understand some details of the structure of the databases it reads. Axiom has 5 databases, the interp.daase, operation.daase, category.daase, compress.daase, and browse.daase. The compress.daase is special and does not follow the normal database format.

65.1.1 kaf File Format

This documentation refers to kaf files which are random access files. nrlib files are kaf files (look for nrlib/index.kaf) The format of a random access file is

```
byte-offset-of-key-table
first-entry
second-entry
...
last-entry
((key1 . first-entry-byte-address)
 (key2 . second-entry-byte-address)
 ...
 (keyN . last-entry-byte-address))
```

The key table is a standard lisp alist.

To open a database you fetch the first number, seek to that location, and (read) which returns the key-data alist. To look up data you index into the key-data alist, find the ith-entry-byte-address, seek to that address, and (read).

For instance, see src/share/algebra/users.daase/index.kaf

One existing optimization is that if the data is a simple thing like a symbol then the nth-entry-byte-address is replaced by immediate data.

Another existing one is a compression algorithm applied to the data so that the very long names don't take up so much space. We could probably remove the compression algorithm as 64k is no longer considered 'huge'. The database-abbreviation routine handles this on read and write-compress handles this on write. The squeeze routine is used to compress the keys, the unsqueeze routine uncompresses them. Making these two routines disappear should remove all of the compression.

Indeed, a faster optimization is to simply read the whole database into the image before it is saved. The system would be easier to understand and the interpreter would be faster.

The fastest optimization is to fix the time stamp mechanism which is currently broken. Making this work requires a small bit of coordination at 'make' time which I forgot to implement.

65.1.2 Database Files

Database files are very similar to kaf files except that there is an optimization (currently broken) which makes the first item a pair of two numbers. The first number in the pair is the offset of the key-value table, the second is a time stamp. If the time stamp in the database matches the time stamp in the image the database is not needed (since the internal hash tables already contain all of the information). When the database is built the time stamp is saved in both the gcl image and the database.

Regarding the 'ancestors field for a category: At database build time there exists a *ancestors-hash* hash table that gets filled with CATEGORY (not domain) ancestor information. This later provides the information that goes into interp.daase This *ancestors-hash* does not exist at normal runtime (it can be made by a call to genCategoryTable). Note that the ancestor information in *ancestors-hash* (and hence interp.daase) involves `#1`, `#2`, etc instead of R, Coef, etc. The latter thingies appear in all .nrlib/index.kaf files. So we need to be careful when we)lib categories and update the ancestor info.

This file contains the code to build, open and access the .daase files. This file contains the code to)library nrlibs and asy files

There is a major issue about the data that resides in these databases. the fundamental problem is that the system requires more information to build the databases than it needs to run the interpreter. in particular, modemap.daase is constructed using properties like "modemaps" but the interpreter will never ask for this information.

So, the design is as follows:

- the modemap.daase needs to be built. this is done by doing a)library on ALL of the nrlib files that are going into the system. this will bring in "modemap" information and add it to the *modemaps-hash* hashtable.
- database build proceeds, accessing the "modemap" property from the hashtables. once this completes this information is never used again.
- the interp.daase database is built. this contains only the information necessary to run the interpreter. note that during the running of the interpreter users can extend the

system by do a)library on a new nrlib file. this will cause fields such as "modemap" to be read and hashed.

Each constructor (e.g. LIST) had one library directory (e.g. LIST.nrlib). This directory contained a random access file called the index.kaf file. These files contain runtime information such as the operationAlist and the ConstructorModemap. At system build time we merge all of these .nrlib/index.kaf files into one database, INTERP.daase. Requests to get information from this database are cached so that multiple references do not cause additional disk i/o.

This database is left open at all times as it is used frequently by the interpreter. one minor complication is that newly compiled files need to override information that exists in this database.

The design calls for constructing a random read (kaf format) file that is accessed by functions that cache their results. when the database is opened the list of constructor-index pairs is hashed by constructor name. a request for information about a constructor causes the information to replace the index in the hash table. since the index is a number and the data is a non-numeric sexpr there is no source of confusion about when the data needs to be read.

The format of this new database is as follows:

```
first entry:
  an integer giving the byte offset to the constructor alist
  at the bottom of the file
second and subsequent entries (one per constructor)
  (operationAlist)
  (constructorModemap)
  ....
last entry: (pointed at by the first entry)
  an alist of (constructor . index) e.g.
     ( (PI offset-of-operationAlist offset-of-constructorModemap)
     (NNI offset-of-operationAlist offset-of-constructorModemap)
      ....)
 This list is read at open time and hashed by the car of each item.
```

The system has been changed to use the property list of the symbols rather than hash tables. since we already hashed once to get the symbol we need only an offset to get the property list. this also has the advantage that eq hash tables no longer need to be moved during garbage collection.

There are 3 potential speedups that could be done.

- the best would be to use the value cell of the symbol rather than the property list but i'm unable to determine all uses of the value cell at the present time.
- a second speedup is to guarantee that the property list is a single item, namely the database structure. this removes an assoc but leaves one open to breaking the system if someone adds something to the property list. this was not done because of the danger mentioned.

- a third speedup is to make the getdatabase call go away, either by making it a macro or eliding it entirely. this was not done because we want to keep the flexibility of changing the database forms.

The new design does not use hash tables. the database structure contains an entry for each item that used to be in a hash table. initially the structure contains file-position pointers and these are replaced by real data when they are first looked up. the database structure is kept on the property list of the constructor, thus, (get '—DenavitHartenbergMatrix— 'database) will return the database structure object.

Each operation has a property on its symbol name called 'operation which is a list of all of the signatures of operations with that name.

65.1.3 defstruct $database

— initvars —

```
(defstruct database
 abbreviation               ; interp.
 ancestors                  ; interp.
 constructor                ; interp.
 constructorcategory        ; interp.
 constructorkind            ; interp.
 constructormodemap         ; interp.
 cosig                      ; interp.
 defaultdomain              ; interp.
 modemaps                   ; interp.
 niladic                    ; interp.
 object                     ; interp.
 operationalist             ; interp.
 documentation              ; browse.
 constructorform            ; browse.
 attributes                 ; browse.
 predicates                 ; browse.
 sourcefile                 ; browse.
 parents                    ; browse.
 users                      ; browse.
 dependents                 ; browse.
 spare                      ; superstition
 ) ; database structure
```

————-

65.1.4 defvar $*defaultdomain-list*

There are only a small number of domains that have default domains. rather than keep this slot in every domain we maintain a list here.

— **initvars** —

```
(defvar *defaultdomain-list* '(
  (|MultisetAggregate| |Multiset|)
  (|FunctionSpace| |Expression|)
  (|AlgebraicallyClosedFunctionSpace| |Expression|)
  (|ThreeSpaceCategory| |ThreeSpace|)
  (|DequeueAggregate| |Dequeue|)
  (|ComplexCategory| |Complex|)
  (|LazyStreamAggregate| |Stream|)
  (|AssociationListAggregate| |AssociationList|)
  (|QuaternionCategory| |Quaternion|)
  (|PriorityQueueAggregate| |Heap|)
  (|PointCategory| |Point|)
  (|PlottableSpaceCurveCategory| |Plot3D|)
  (|PermutationCategory| |Permutation|)
  (|StringCategory| |String|)
  (|FileNameCategory| |FileName|)
  (|OctonionCategory| |Octonion|)))
```

———-

65.1.5 defvar $*operation-hash*

— **initvars** —

```
(defvar *operation-hash* nil "given an operation name, what are its modemaps?")
```

———-

65.1.6 defvar $*hasCategory-hash*

This hash table is used to answer the question"does domain x have category y?". this is answered by constructing a pair of (x . y) and doing an equal hash into this table.

— **initvars** —

```
(defvar *hasCategory-hash* nil "answers x has y category questions")
```

———-

65.1.7 defvar $*miss*

This variable is used for debugging. If a hash table lookup fails and this variable is non-nil then a message is printed.

— **initvars** —

```
(defvar *miss* nil "print out cache misses on getdatabase calls")
```

————-

Note that constructorcategory information need only be kept for items of type category. this will be fixed in the next iteration when the need for the various caches are reviewed

Note that the *modemaps-hash* information does not need to be kept for system files. these are precomputed and kept in modemap.daase however, for user-defined files these are needed. Currently these are added to the database for 2 reasons; there is a still-unresolved issue of user database extensions and this information is used during database build time

65.1.8 Database streams

This are the streams for the databases. They are always open. There is an optimization for speeding up system startup. If the database is opened and the ..-stream-stamp* variable matches the position information in the database then the database is NOT read in and is assumed to match the in-core version

65.1.9 defvar $*compressvector*

— **initvars** —

```
(defvar *compressvector* nil "a vector of things to compress in the databases")
```

————-

65.1.10 defvar $*compressVectorLength*

— **initvars** —

```
(defvar *compressVectorLength* 0 "length of the compress vector")
```

————-

65.1.11 defvar $*compress-stream*

— **initvars** —

```
(defvar *compress-stream* nil "an stream containing the compress vector")
```

———-

65.1.12 defvar $*compress-stream-stamp*

— **initvars** —

```
(defvar *compress-stream-stamp* 0 "*compress-stream* (position . time)")
```

———-

65.1.13 defvar $*interp-stream*

— **initvars** —

```
(defvar *interp-stream* nil "an open stream to the interpreter database")
```

———-

65.1.14 defvar $*interp-stream-stamp*

— **initvars** —

```
(defvar *interp-stream-stamp* 0 "*interp-stream* (position . time)")
```

———-

65.1.15 defvar $*operation-stream*

This is indexed by operation, not constructor

— **initvars** —

```
(defvar *operation-stream* nil "the stream to operation.daase")
```

———-

65.1.16 defvar $*operation-stream-stamp*

— **initvars** —

```
(defvar *operation-stream-stamp* 0 "*operation-stream* (position . time)")
```

———-

65.1.17 defvar $*browse-stream*

— **initvars** —

```
(defvar *browse-stream* nil "an open stream to the browser database")
```

———-

65.1.18 defvar $*browse-stream-stamp*

— **initvars** —

```
(defvar *browse-stream-stamp* 0 "*browse-stream* (position . time)")
```

———-

65.1.19 defvar $*category-stream*

This is indexed by (domain . category)
— **initvars** —

```
(defvar *category-stream* nil "an open stream to the category table")
```

———-

65.1.20 defvar $*category-stream-stamp*

— **initvars** —

```
(defvar *category-stream-stamp* 0 "*category-stream* (position . time)")
```

———-

65.1.21 defvar $*allconstructors*

— **initvars** —

```
(defvar *allconstructors* nil "a list of all the constructors in the system")
```

———-

65.1.22 defvar $*allOperations*

— **initvars** —

```
(defvar *allOperations* nil "a list of all the operations in the system")
```

———-

65.1.23 defun Reset all hash tables before saving system

[compressopen p**??**]
[interpopen p**??**]
[operationopen p**??**]
[browseopen p**??**]
[categoryopen p**??**]
[initial-getdatabase p992]
[*sourcefiles* p**??**]
[*interp-stream* p989]
[*operation-stream* p989]
[*category-stream* p990]
[*browse-stream* p990]
[*category-stream-stamp* p991]
[*operation-stream-stamp* p990]

[*interp-stream-stamp* p989]
[*compress-stream-stamp* p989]
[*compressvector* p988]
[*allconstructors* p991]
[*operation-hash* p987]
[*hascategory-hash* p??]

— **defun resethashtables** —

```
(defun resethashtables ()
 "set all -hash* to clean values. used to clean up core before saving system"
 (declare (special *sourcefiles* *interp-stream* *operation-stream*
                   *category-stream* *browse-stream* *category-stream-stamp*
                   *operation-stream-stamp* *interp-stream-stamp*
                   *compress-stream-stamp* *compressvector*
                   *allconstructors* *operation-hash* *hascategory-hash*))
 (setq *hascategory-hash* (make-hash-table :test #'equal))
 (setq *operation-hash* (make-hash-table))
 (setq *allconstructors* nil)
 (setq *compressvector* nil)
 (setq *sourcefiles* nil)
 (setq *compress-stream-stamp* '(0 . 0))
 (compressopen)
 (setq *interp-stream-stamp* '(0 . 0))
 (interpopen)
 (setq *operation-stream-stamp* '(0 . 0))
 (operationopen)
 (setq *browse-stream-stamp* '(0 . 0))
 (browseopen)
 (setq *category-stream-stamp* '(0 . 0))
 (categoryopen) ;note: this depends on constructorform in browse.daase
 (initial-getdatabase)
 (close *interp-stream*)
 (close *operation-stream*)
 (close *category-stream*)
 (close *browse-stream*)
 (gbc t))
```

———-

65.1.24 defun Preload algebra into saved system

[getdatabase p1001]
[getEnv p??]

— **defun initial-getdatabase** —

```
(defun initial-getdatabase ()
 "fetch data we want in the saved system"
 (let (hascategory constructormodemapAndoperationalist operation constr)
 (format t "Initial getdatabase~%")
 (setq hascategory '(
  (|Equation| . |Ring|)
  (|Expression| . |CoercibleTo|) (|Expression| . |CommutativeRing|)
  (|Expression| . |IntegralDomain|) (|Expression| . |Ring|)
  (|Float| . |RetractableTo|)
  (|Fraction| . |Algebra|) (|Fraction| . |CoercibleTo|)
  (|Fraction| . |OrderedSet|) (|Fraction| . |RetractableTo|)
  (|Integer| . |Algebra|) (|Integer| . |CoercibleTo|)
  (|Integer| . |ConvertibleTo|) (|Integer| . |LinearlyExplicitRingOver|)
  (|Integer| . |RetractableTo|)
  (|List| . |CoercibleTo|) (|List| . |FiniteLinearAggregate|)
  (|List| . |OrderedSet|)
  (|Polynomial| . |CoercibleTo|) (|Polynomial| . |CommutativeRing|)
  (|Polynomial| . |ConvertibleTo|) (|Polynomial| . |OrderedSet|)
  (|Polynomial| . |RetractableTo|)
  (|Symbol| . |CoercibleTo|) (|Symbol| . |ConvertibleTo|)
  (|Variable| . |CoercibleTo|)))
 (dolist (pair hascategory) (getdatabase pair 'hascategory))
 (setq constructormodemapAndoperationalist '(
  |BasicOperator|  |Boolean|
  |CardinalNumber| |Color|  |Complex|
  |Database|
  |Equation| |EquationFunctions2| |Expression|
  |Float| |Fraction| |FractionFunctions2|
  |Integer| |IntegralDomain|
  |Kernel|
  |List|
  |Matrix| |MappingPackage1|
  |Operator| |OutputForm|
  |NonNegativeInteger|
  |ParametricPlaneCurve| |ParametricSpaceCurve| |Point| |Polynomial|
  |PolynomialFunctions2| |PositiveInteger|
  |Ring|
  |SetCategory| |SegmentBinding| |SegmentBindingFunctions2| |DoubleFloat|
  |SparseMultivariatePolynomial| |SparseUnivariatePolynomial| |Segment|
  |String| |Symbol|
  |UniversalSegment|
  |Variable|  |Vector|))
 (dolist (con constructormodemapAndoperationalist)
  (getdatabase con 'constructormodemap)
  (getdatabase con 'operationalist))
 (setq operation '(
  |+| |-| |*| |/| |**| |coerce| |convert| |elt| |equation|
  |float| |sin| |cos| |map| |SEGMENT|))
 (dolist (op operation) (getdatabase op 'operation))
 (setq constr '( ;these are sorted least-to-most freq. delete early ones first
```

```
 |Factored| |SparseUnivariatePolynomialFunctions2| |TableAggregate&|
 |RetractableTo&| |RecursiveAggregate&| |UserDefinedPartialOrdering|
 |None| |UnivariatePolynomialCategoryFunctions2| |IntegerPrimesPackage|
 |SetCategory&| |IndexedExponents| |QuotientFieldCategory&| |Polynomial|
 |EltableAggregate&| |PartialDifferentialRing&| |Set|
 |UnivariatePolynomialCategory&| |FlexibleArray|
 |SparseMultivariatePolynomial| |PolynomialCategory&|
 |DifferentialExtension&| |IndexedFlexibleArray| |AbelianMonoidRing&|
 |FiniteAbelianMonoidRing&| |DivisionRing&| |FullyLinearlyExplicitRingOver&|
 |IndexedVector| |IndexedOneDimensionalArray| |LocalAlgebra| |Localize|
 |Boolean| |Field&| |Vector| |IndexedDirectProductObject| |Aggregate&|
 |PolynomialRing| |FreeModule| |IndexedDirectProductAbelianGroup|
 |IndexedDirectProductAbelianMonoid| |SingletonAsOrderedSet|
 |SparseUnivariatePolynomial| |Fraction| |Collection&| |HomogeneousAggregate&|
 |RepeatedSquaring| |IntegerNumberSystem&| |AbelianSemiGroup&|
 |AssociationList| |OrderedRing&| |SemiGroup&| |Symbol|
 |UniqueFactorizationDomain&| |EuclideanDomain&| |IndexedAggregate&|
 |GcdDomain&| |IntegralDomain&| |DifferentialRing&| |Monoid&| |Reference|
 |UnaryRecursiveAggregate&| |OrderedSet&| |AbelianGroup&| |Algebra&|
 |Module&| |Ring&| |StringAggregate&| |AbelianMonoid&|
 |ExtensibleLinearAggregate&| |PositiveInteger| |StreamAggregate&|
 |IndexedString| |IndexedList| |ListAggregate&| |LinearAggregate&|
 |Character| |String| |NonNegativeInteger| |SingleInteger|
 |OneDimensionalArrayAggregate&| |FiniteLinearAggregate&| |PrimitiveArray|
 |Integer| |List| |OutputForm|))
(dolist (con constr)
 (let ((c (concatenate 'string
            (|getEnv| "AXIOM") "/algebra/"
            (string (getdatabase con 'abbreviation)) ".o")))
   (format t "   preloading ~a.." c)
   (if (probe-file c)
    (progn
     (put con 'loaded c)
     (load c)
     (format t "loaded.~%"))
    (format t "skipped.~%"))))
(format t "~%")))
```

———-

65.1.25 defun Open the interp database

Format of an entry in interp.daase:

```
 (constructor-name
   operationalist
   constructormodemap
   modemaps            -- this should not be needed. eliminate it.
```

```
    object              -- the name of the object file to load for this con.
    constructorcategory -- note that this info is the cadar of the
          constructormodemap for domains and packages so it is stored
          as NIL for them. it is valid for categories.
    niladic             -- t or nil directly
    unused
    cosig                -- kept directly
    constructorkind      -- kept directly
    defaultdomain        -- a short list, for %i
    ancestors            -- used to compute new category updates
  )
```

[unsqueeze p1018]
[make-database p**??**]
[DaaseName p1014]
[$spadroot p12]
[*allconstructors* p991]
[*interp-stream* p989]
[*interp-stream-stamp* p989]

— **defun interpOpen** —

```
(defun interpOpen ()
 "open the interpreter database and hash the keys"
 (declare (special $spadroot *allconstructors* *interp-stream*
                   *interp-stream-stamp*))
 (let (constructors pos stamp dbstruct)
  (setq *interp-stream* (open (DaaseName "interp.daase" nil)))
  (setq stamp (read *interp-stream*))
  (unless (equal stamp *interp-stream-stamp*)
   (format t "   Re-reading interp.daase")
   (setq *interp-stream-stamp* stamp)
   (setq pos (car stamp))
   (file-position *interp-stream* pos)
   (setq constructors (read *interp-stream*))
   (dolist (item constructors)
    (setq item (unsqueeze item))
    (setq *allconstructors* (adjoin (first item) *allconstructors*))
    (setq dbstruct (make-database))
    (setf (get (car item) 'database) dbstruct)
    (setf (database-operationalist dbstruct) (second item))
    (setf (database-constructormodemap dbstruct) (third item))
    (setf (database-modemaps dbstruct) (fourth item))
    (setf (database-object dbstruct) (fifth item))
    (setf (database-constructorcategory dbstruct) (sixth item))
    (setf (database-niladic dbstruct) (seventh item))
    (setf (database-abbreviation dbstruct) (eighth item))
    (setf (get (eighth item) 'abbreviationfor) (first item)) ;invert
    (setf (database-cosig dbstruct) (ninth item))
```

```
    (setf (database-constructorkind dbstruct) (tenth item))
    (setf (database-ancestors dbstruct) (nth 11 item))))
  (format t "~&")))
```

———-

This is an initialization function for the constructor database it sets up 2 hash tables, opens the database and hashes the index values.

There is a slight asymmetry in this code. The sourcefile information for system files is only the filename and extension. For user files it contains the full pathname. when the database is first opened the sourcefile slot contains system names. The lookup function has to prefix the "$spadroot" information if the directory-namestring is null (we don't know the real root at database build time).

An object-hash table is set up to look up nrlib and ao information. this slot is empty until a user does a)library call. We remember the location of the nrlib or ao file for the users local library at that time. A `NIL` result from this probe means that the library is in the system-specified place. When we get into multiple library locations this will also contain system files.

65.1.26 defun Open the browse database

Format of an entry in browse.daase:

```
( constructorname
    sourcefile
    constructorform
    documentation
    attributes
    predicates
)
```

[unsqueeze p1018]
[$spadroot p12]
[*allconstructors* p991]
[*browse-stream* p990]
[*browse-stream-stamp* p990]

— defun browseOpen —

```
(defun browseOpen ()
 "open the constructor database and hash the keys"
 (declare (special $spadroot *allconstructors* *browse-stream*
                   *browse-stream-stamp*))
 (let (constructors pos stamp dbstruct)
  (setq *browse-stream* (open (DaaseName "browse.daase" nil)))
```

```
  (setq stamp (read *browse-stream*))
  (unless (equal stamp *browse-stream-stamp*)
   (format t "   Re-reading browse.daase")
   (setq *browse-stream-stamp* stamp)
   (setq pos (car stamp))
   (file-position *browse-stream* pos)
   (setq constructors (read *browse-stream*))
   (dolist (item constructors)
    (setq item (unsqueeze item))
    (unless (setq dbstruct (get (car item) 'database))
     (format t "browseOpen:~%")
     (format t "the browse database contains a contructor ~a~%" item)
     (format t "that is not in the interp.daase file. we cannot~%")
     (format t "get the database structure for this constructor and~%")
     (warn "will create a new one~%")
     (setf (get (car item) 'database) (setq dbstruct (make-database)))
     (setq *allconstructors* (adjoin item *allconstructors*)))
    (setf (database-sourcefile dbstruct) (second item))
    (setf (database-constructorform dbstruct) (third item))
    (setf (database-documentation dbstruct) (fourth item))
    (setf (database-attributes dbstruct) (fifth item))
    (setf (database-predicates dbstruct) (sixth item))
    (setf (database-parents dbstruct) (seventh item))))
  (format t "~&")))
```

———-

65.1.27 defun Open the category database

[unsqueeze p1018]
[$spadroot p12]
[*hasCategory-hash* p987]
[*category-stream* p990]
[*category-stream-stamp* p991]

— **defun categoryOpen** —

```
(defun categoryOpen ()
 "open category.daase and hash the keys"
 (declare (special $spadroot *hasCategory-hash* *category-stream*
                   *category-stream-stamp*))
 (let (pos keys stamp)
  (setq *category-stream* (open (DaaseName "category.daase" nil)))
  (setq stamp (read *category-stream*))
  (unless (equal stamp *category-stream-stamp*)
   (format t "   Re-reading category.daase")
   (setq *category-stream-stamp* stamp)
```

```
   (setq pos (car stamp))
   (file-position *category-stream* pos)
   (setq keys (read *category-stream*))
   (setq *hasCategory-hash* (make-hash-table :test #'equal))
   (dolist (item keys)
    (setq item (unsqueeze item))
    (setf (gethash (first item) *hasCategory-hash*) (second item))))
  (format t "~&")))
```

———-

65.1.28 defun Open the operations database

[unsqueeze p1018]
[$spadroot p12]
[*operation-hash* p987]
[*operation-stream* p989]
[*operation-stream-stamp* p990]

— **defun operationOpen** —

```
(defun operationOpen ()
 "read operation database and hash the keys"
 (declare (special $spadroot *operation-hash* *operation-stream*
                   *operation-stream-stamp*))
 (let (operations pos stamp)
  (setq *operation-stream* (open (DaaseName "operation.daase" nil)))
  (setq stamp (read *operation-stream*))
  (unless (equal stamp *operation-stream-stamp*)
   (format t "   Re-reading operation.daase")
   (setq *operation-stream-stamp* stamp)
   (setq pos (car stamp))
   (file-position *operation-stream* pos)
   (setq operations (read *operation-stream*))
   (dolist (item operations)
    (setq item (unsqueeze item))
    (setf (gethash (car item) *operation-hash*) (cdr item))))
  (format t "~&")))
```

———-

65.1.29 defun Add operations from newly compiled code

[getdatabase p1001]
[*operation-hash* p987]

— **defun addoperations** —

```
(defun addoperations (constructor oldmaps)
 "add ops from a )library domain to *operation-hash*"
 (declare (special *operation-hash*))
 (dolist (map oldmaps) ; out with the old
  (let (oldop op)
   (setq op (car map))
   (setq oldop (getdatabase op 'operation))
   (setq oldop (lisp::delete (cdr map) oldop :test #'equal))
   (setf (gethash op *operation-hash*) oldop)))
 (dolist (map (getdatabase constructor 'modemaps)) ; in with the new
  (let (op newmap)
   (setq op (car map))
   (setq newmap (getdatabase op 'operation))
   (setf (gethash op *operation-hash*) (cons (cdr map) newmap)))))
```

———-

65.1.30 defun Show all database attributes of a constructor

[getdatabase p1001]

— **defun showdatabase** —

```
(defun showdatabase (constructor)
 (format t "~&~a: ~a~%" 'constructorkind
  (getdatabase constructor 'constructorkind))
 (format t "~&~a: ~a~%" 'cosig
  (getdatabase constructor 'cosig))
 (format t "~&~a: ~a~%" 'operation
  (getdatabase constructor 'operation))
 (format t "~&~a: ~%" 'constructormodemap)
  (pprint (getdatabase constructor 'constructormodemap))
 (format t "~&~a: ~%" 'constructorcategory)
  (pprint (getdatabase constructor 'constructorcategory))
 (format t "~&~a: ~%" 'operationalist)
  (pprint (getdatabase constructor 'operationalist))
 (format t "~&~a: ~%" 'modemaps)
  (pprint (getdatabase constructor 'modemaps))
 (format t "~&~a: ~a~%" 'hascategory
  (getdatabase constructor 'hascategory))
 (format t "~&~a: ~a~%" 'object
  (getdatabase constructor 'object))
 (format t "~&~a: ~a~%" 'niladic
  (getdatabase constructor 'niladic))
```

```
(format t "~&~a: ~a~%" 'abbreviation
 (getdatabase constructor 'abbreviation))
(format t "~&~a: ~a~%" 'constructor?
 (getdatabase constructor 'constructor?))
(format t "~&~a: ~a~%" 'constructor
 (getdatabase constructor 'constructor))
(format t "~&~a: ~a~%" 'defaultdomain
 (getdatabase constructor 'defaultdomain))
(format t "~&~a: ~a~%" 'ancestors
 (getdatabase constructor 'ancestors))
(format t "~&~a: ~a~%" 'sourcefile
 (getdatabase constructor 'sourcefile))
(format t "~&~a: ~a~%" 'constructorform
 (getdatabase constructor 'constructorform))
(format t "~&~a: ~a~%" 'constructorargs
 (getdatabase constructor 'constructorargs))
(format t "~&~a: ~a~%" 'attributes
 (getdatabase constructor 'attributes))
(format t "~&~a: ~%" 'predicates)
 (pprint (getdatabase constructor 'predicates))
(format t "~&~a: ~a~%" 'documentation
 (getdatabase constructor 'documentation))
(format t "~&~a: ~a~%" 'parents
 (getdatabase constructor 'parents)))
```

———-

65.1.31 defun Set a value for a constructor key in the database

[make-database p**??**]

— defun setdatabase —

```
(defun setdatabase (constructor key value)
 (let (struct)
  (when (symbolp constructor)
   (unless (setq struct (get constructor 'database))
    (setq struct (make-database))
    (setf (get constructor 'database) struct))
   (case key
    (abbreviation
     (setf (database-abbreviation struct) value)
     (when (symbolp value)
      (setf (get value 'abbreviationfor) constructor)))
    (constructorkind
     (setf (database-constructorkind struct) value))))))
```

———-

65.1.32 defun Delete a value for a constructor key in the database

— **defun deldatabase** —

```
(defun deldatabase (constructor key)
  (when (symbolp constructor)
   (case key
    (abbreviation
     (setf (get constructor 'abbreviationfor) nil)))))
```

———-

65.1.33 defun Get constructor information for a database key

[warn p**??**]
[unsqueeze p1018]
[$spadroot p12]
[*miss* p988]
[*hascategory-hash* p**??**]
[*operation-hash* p987]
[*browse-stream* p990]
[*defaultdomain-list* p987]
[*interp-stream* p989]
[*category-stream* p990]
[*hasCategory-hash* p987]
[*operation-stream* p989]

— **defun getdatabase** —

```
(defun getdatabase (constructor key)
 (declare (special $spadroot) (special *miss*))
 (when (eq *miss* t) (format t "getdatabase call: ~20a ~a~%" constructor key))
 (let (data table stream ignore struct)
  (declare (ignore ignore)
           (special *hascategory-hash* *operation-hash*
                    *browse-stream* *defaultdomain-list* *interp-stream*
                    *category-stream* *hasCategory-hash* *operation-stream*))
  (when (or (symbolp constructor)
            (and (eq key 'hascategory) (consp constructor)))
  (case key
; note that abbreviation, constructorkind and cosig are heavy hitters
; thus they occur first in the list of things to check
```

```
(abbreviation
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
   (setq data (database-abbreviation struct))))
(constructorkind
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-constructorkind struct))))
(cosig
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-cosig struct))))
(operation
 (setq stream *operation-stream*)
 (setq data (gethash constructor *operation-hash*)))
(constructormodemap
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-constructormodemap struct))))
(constructorcategory
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-constructorcategory struct))
  (when (null data) ;domain or package then subfield of constructormodemap
   (setq data (cadar (getdatabase constructor 'constructormodemap))))))
(operationalist
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-operationalist struct))))
(modemaps
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-modemaps struct))))
(hascategory
 (setq table  *hasCategory-hash*)
 (setq stream *category-stream*)
 (setq data (gethash constructor table)))
(object
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-object struct))))
(asharp?
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-object struct))))
(niladic
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-niladic struct))))
(constructor?
```

```
 (when (setq struct (get constructor 'database))
   (setq data (when (database-operationalist struct) t))))
(superdomain ; only 2 superdomains in the world
 (case constructor
  (|NonNegativeInteger|
   (setq data '((|Integer|) (IF (< |#1| 0) |false| |true|))))
  (|PositiveInteger|
   (setq data '((|NonNegativeInteger|) (< 0 |#1|))))))
(constructor
 (when (setq data (get constructor 'abbreviationfor))))
(defaultdomain
 (setq data (cadr (assoc constructor *defaultdomain-list*))))
(ancestors
 (setq stream *interp-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-ancestors struct))))
(sourcefile
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-sourcefile struct))))
(constructorform
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-constructorform struct))))
(constructorargs
 (setq data (cdr (getdatabase constructor 'constructorform))))
(attributes
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-attributes struct))))
(predicates
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-predicates struct))))
(documentation
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-documentation struct))))
(parents
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-parents struct))))
(users
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-users struct))))
(dependents
 (setq stream *browse-stream*)
 (when (setq struct (get constructor 'database))
  (setq data (database-dependents struct))))
```

```
 (otherwise  (warn "~%(GETDATABASE ~a ~a) failed~%" constructor key)))
(when (numberp data)      ;fetch the real data
 (when *miss* (format t "getdatabase miss: ~20a ~a~%" constructor key))
 (file-position stream data)
 (setq data (unsqueeze (read stream)))
 (case key ; cache the result of the database read
  (operation           (setf (gethash constructor *operation-hash*) data))
  (hascategory         (setf (gethash constructor *hascategory-hash*) data))
  (constructorkind     (setf (database-constructorkind struct) data))
  (cosig               (setf (database-cosig struct) data))
  (constructormodemap  (setf (database-constructormodemap struct) data))
  (constructorcategory (setf (database-constructorcategory struct) data))
  (operationalist      (setf (database-operationalist struct) data))
  (modemaps            (setf (database-modemaps struct) data))
  (object              (setf (database-object struct) data))
  (niladic             (setf (database-niladic struct) data))
  (abbreviation        (setf (database-abbreviation struct) data))
  (constructor         (setf (database-constructor struct) data))
  (ancestors           (setf (database-ancestors struct) data))
  (constructorform     (setf (database-constructorform struct) data))
  (attributes          (setf (database-attributes struct) data))
  (predicates          (setf (database-predicates struct) data))
  (documentation       (setf (database-documentation struct) data))
  (parents             (setf (database-parents struct) data))
  (users               (setf (database-users struct) data))
  (dependents          (setf (database-dependents struct) data))
  (sourcefile          (setf (database-sourcefile struct) data))))
 (case key ; fixup the special cases
  (sourcefile
   (when (and data (string= (directory-namestring data) "")
           (string= (pathname-type data) "spad"))
    (setq data
     (concatenate 'string $spadroot "/../../src/algebra/" data))))
  (asharp?                                ; is this asharp code?
   (if (consp data)
    (setq data (cdr data))
    (setq data nil)))
  (object                                 ; fix up system object pathname
   (if (consp data)
     (setq data
           (if (string= (directory-namestring (car data)) "")
               (concatenate 'string $spadroot "/algebra/" (car data) ".o")
             (car data)))
    (when (and data (string= (directory-namestring data) ""))
     (setq data (concatenate 'string $spadroot "/algebra/" data ".o")))))))
data))
```

———-

65.1.34 defun The)library top level command

[localdatabase p1005]
[extendLocalLibdb p??]
[tersyscommand p448]
[$newConlist p??]
[$options p??]

— **defun library** —

```
(defun |library| (args)
 (let (original-directory)
 (declare (special |$options| |$newConlist|))
  (setq original-directory (get-current-directory))
  (setq |$newConlist| nil)
  (localdatabase args |$options|)
  (|extendLocalLibdb| |$newConlist|)
  (system::chdir original-directory)
  (tersyscommand)))
```

———-

65.1.35 defun Read a local filename and update the hash tables

The localdatabase function tries to find files in the order of:

- nrlib/index.kaf
- .asy
- .ao,
- asharp to .asy

[sayKeyedMsg p327]
[localnrlib p1007]
[$forceDatabaseUpdate p??]
[$ConstructorCache p??]
[*index-filename* p??]

— **defun localdatabase** —

```
(defun localdatabase (filelist options &optional (make-database? nil))
 "read a local filename and update the hash tables"
 (labels (
  (processOptions (options)
   (let (only dir noexpose)
```

```
   (when (setq only (assoc '|only| options))
    (setq options (lisp::delete only options :test #'equal))
    (setq only (cdr only)))
   (when (setq dir (assoc '|dir| options))
    (setq options (lisp::delete dir options :test #'equal))
    (setq dir (second dir))
    (when (null dir)
     (|sayKeyedMsg| 'S2IU0002 nil) ))
   (when (setq noexpose (assoc '|noexpose| options))
    (setq options (lisp::delete noexpose options :test #'equal))
    (setq noexpose 't) )
   (when options
    (format t "   Ignoring unknown )library option: ~a~%" options))
   (values only dir noexpose)))
 (processDir (dirarg thisdir)
  (let (allfiles)
   (declare (special vmlisp::*index-filename*))
   (system:chdir (string dirarg))
   (setq allfiles (directory "*"))
   (system:chdir thisdir)
    (mapcan #'(lambda (f)
     (when (string-equal (pathname-type f) "nrlib")
      (list (concatenate 'string (namestring f) "/"
                         vmlisp::*index-filename*)))) allfiles))))
(let (thisdir nrlibs object only dir key  (|$forceDatabaseUpdate| t) noexpose)
 (declare (special |$forceDatabaseUpdate| vmlisp::*index-filename*
                   |$ConstructorCache|))
 (setq thisdir (namestring (truename ".")))
 (setq noexpose nil)
 (multiple-value-setq (only dir noexpose) (processOptions options))
    ;don't force exposure during database build
 (if make-database? (setq noexpose t))
 (when dir (setq nrlibs (processDir dir thisdir)))
 (dolist (file filelist)
  (let ((filename (pathname-name file))
        (namedir (directory-namestring file)))
   (unless namedir (setq thisdir (concatenate 'string thisdir "/")))
   (cond
    ((setq file (probe-file
      (concatenate 'string namedir filename ".nrlib/"
                   vmlisp::*index-filename*)))
     (push (namestring file) nrlibs))
    ('else (format t "   )library cannot find the file ~a.~%" filename)))))
 (dolist (file (nreverse nrlibs))
  (setq key (pathname-name (first (last (pathname-directory file)))))
  (setq object (concatenate 'string (directory-namestring file) "code"))
  (localnrlib key file object make-database? noexpose))
 (clrhash |$ConstructorCache|))))
```

———-

65.1.36 defun Update the database from an nrlib index.kaf file

[getdatabase p1001]
[make-database p**??**]
[addoperations p998]
[sublislis p**??**]
[updateDatabase p1009]
[installConstructor p**??**]
[updateCategoryTable p**??**]
[categoryForm? p**??**]
[setExposeAddConstr p692]
[startTimingProcess p**??**]
[loadLibNoUpdate p1031]
[sayKeyedMsg p327]
[$FormalMapVariableList p**??**]
[*allOperations* p991]
[*allconstructors* p991]

— defun localnrlib —

```
(defun localnrlib (key nrlib object make-database? noexpose)
 "given a string pathname of an index.kaf and the object update the database"
 (labels (
  (fetchdata (alist in index)
   (let (pos)
    (setq pos (third (assoc index alist :test #'string=)))
    (when pos
     (file-position in pos)
     (read in)))))
 (let (alist kind (systemdir? nil) pos constructorform oldmaps abbrev dbstruct)
  (declare (special *allOperations* *allconstructors*
                    |$FormalMapVariableList|))
  (with-open-file (in nrlib)
   (file-position in (read in))
   (setq alist (read in))
   (setq pos (third (assoc "constructorForm" alist :test #'string=)))
   (file-position in pos)
   (setq constructorform (read in))
   (setq key (car constructorform))
   (setq oldmaps (getdatabase key 'modemaps))
   (setq dbstruct (make-database))
   (setq *allconstructors* (adjoin key *allconstructors*))
   (setf (get key 'database) dbstruct) ; store the struct, side-effect it...
   (setf (database-constructorform dbstruct) constructorform)
   (setq *allOperations* nil)   ; force this to recompute
```

```
 (setf (database-object dbstruct) object)
 (setq abbrev
   (intern (pathname-name (first (last (pathname-directory object))))))
 (setf (database-abbreviation dbstruct) abbrev)
 (setf (get abbrev 'abbreviationfor) key)
 (setf (database-operationalist dbstruct) nil)
 (setf (database-operationalist dbstruct)
  (fetchdata alist in "operationAlist"))
 (setf (database-constructormodemap dbstruct)
  (fetchdata alist in "constructorModemap"))
 (setf (database-modemaps dbstruct) (fetchdata alist in "modemaps"))
 (setf (database-sourcefile dbstruct) (fetchdata alist in "sourceFile"))
 (when make-database?
  (setf (database-sourcefile dbstruct)
   (file-namestring  (database-sourcefile dbstruct))))
 (setf (database-constructorkind dbstruct)
  (setq kind (fetchdata alist in "constructorKind")))
 (setf (database-constructorcategory dbstruct)
  (fetchdata alist in "constructorCategory"))
 (setf (database-documentation dbstruct)
  (fetchdata alist in "documentation"))
 (setf (database-attributes dbstruct)
  (fetchdata alist in "attributes"))
 (setf (database-predicates dbstruct)
  (fetchdata alist in "predicates"))
 (setf (database-niladic dbstruct)
  (when (fetchdata alist in "NILADIC") t))
(addoperations key oldmaps)
(unless make-database?
 (if (eq kind '|category|)
  (setf (database-ancestors dbstruct)
   (sublislis |$FormalMapVariableList|
    (cdr constructorform) (fetchdata alist in "ancestors"))))
 (|updateDatabase| key key systemdir?) ;makes many hashtables???
 (|installConstructor| key kind) ;used to be key cname ...
 (|updateCategoryTable| key kind)
 (if |$InteractiveMode| (setq |$CategoryFrame| |$EmptyEnvironment|)))
(setf (database-cosig dbstruct)
  (cons nil (mapcar #'|categoryForm?|
   (cddar (database-constructormodemap dbstruct)))))
(remprop key 'loaded)
(if (null noexpose) (|setExposeAddConstr| (cons key nil)))
(setf (symbol-function key) ; sets the autoload property for cname
  #'(lambda (&rest args)
   (unless (get key 'loaded)
    (|startTimingProcess| '|load|)
    (|loadLibNoUpdate| key key object)) ; used to be cname key
   (apply key args)))
(|sayKeyedMsg| 'S2IU0001 (list key object))))))
```

———-

65.1.37 defun updateDatabase

For now in NRUNTIME do database update only if forced [constructor? p**??**]
[clearClams p**??**]
[clearAllSlams p**??**]
[$forceDatabaseUpdate p**??**]

— **defun updateDatabase** —

```
(defun |updateDatabase| (fname cname systemdirp)
 (declare (ignore fname))
 (declare (special |$forceDatabaseUpdate|))
  (when |$forceDatabaseUpdate|
   (when (|constructor?| cname)
    (|clearClams|)
    (|clearAllSlams| nil)
    (when (getl cname 'loaded) (|clearConstructorCaches|)))
   (when (or |$forceDatabaseUpdate| (null systemdirp))
    (|clearClams|)
    (|clearAllSlams| nil))))
```

———-

65.1.38 defun Make new databases

Making new databases consists of:

1. reset all of the system hash tables
2. set up Union, Record and Mapping
3. map)library across all of the system files (fills the databases)
4. loading some normally autoloaded files
5. making some database entries that are computed (like ancestors)
6. writing out the databases
7. write out 'warm' data to be loaded into the image at build time

Note that this process should be done in a clean image followed by a rebuild of the system image to include the new index pointers (e.g. *interp-stream-stamp*)

The system will work without a rebuild but it needs to re-read the databases on startup. Rebuilding the system will cache the information into the image and the databases are opened

but not read, saving considerable startup time. Also note that the order the databases are written out is critical. The interp.daase depends on prior computations and has to be written out last.

The build-name-to-pamphlet-hash builds a hash table whose key-¿value is:

- abbreviation -¿ pamphlet file name
- abbreviation-line -¿ pamphlet file position
- constructor -¿ pamphlet file name
- constructor-line -¿ pamphlet file position

is the symbol of the constructor name and whose value is the name of the source file without any path information. We hash the constructor abbreviation to pamphlet file name. [local-database p1005]
[getEnv p**??**]
[browserAutoloadOnceTrigger p**??**]
[mkTopicHashTable p**??**]
[buildLibdb p**??**]
[dbSplitLibdb p**??**]
[mkUsersHashTable p**??**]
[saveUsersHashTable p1014]
[mkDependentsHashTable p**??**]
[saveDependentsHashTable p1013]
[write-compress p1016]
[write-browsedb p1024]
[write-operationdb p1026]
[write-categorydb p1025]
[allConstructors p1027]
[categoryForm? p**??**]
[domainsOf p**??**]
[getConstructorForm p**??**]
[write-interpdb p1022]
[write-warmdata p1027]
[$constructorList p**??**]
[*sourcefiles* p**??**]
[*compressvector* p988]
[*allconstructors* p991]
[*operation-hash* p987]

— defun make-databases —

```
(defun make-databases (ext dirlist)
 (labels (
  (build-name-to-pamphlet-hash (dir)
   (let ((ht (make-hash-table)) (eof '(done)) point mark abbrev name file ns)
```

```
   (dolist (fn (directory dir))
    (with-open-file (f fn)
     (do ((ln (read-line f nil eof) (read-line f nil eof))
          (line 0 (incf line)))
         ((eq ln eof))
     (when (and (setq mark (search ")abb" ln)) (= mark 0))
       (setq mark (position #\space ln :from-end t))
       (setq name (intern (string-trim '(#\space) (subseq ln mark))))
       (cond
         ((setq mark (search "domain" ln)) (setq mark (+ mark 7)))
         ((setq mark (search "package" ln)) (setq mark (+ mark 8)))
         ((setq mark (search "category" ln)) (setq mark (+ mark 9))))
       (setq point (position #\space ln :start (+ mark 1)))
       (setq abbrev
        (intern (string-trim '(#\space) (subseq ln mark point))))
       (setq ns (namestring fn))
       (setq mark (position #\/ ns :from-end t))
       (setq file (subseq ns (+ mark 1)))
       (setf (gethash abbrev ht) file)
       (setf (gethash (format nil "~a-line" abbrev) ht) line)
       (setf (gethash name ht) file)
       (setf (gethash (format nil "~a-line" name) ht) line)))))
  ht))
  ;; these are types which have no library object associated with them.
  ;; we store some constructed data to make them perform like library
  ;; objects, the *operationalist-hash* key entry is used by allConstructors
(withSpecialConstructors ()
 (declare (special *allconstructors*))
 ; note: if item is not in *operationalist-hash* it will not be written
 ; Category
 (setf (get '|Category| 'database)
   (make-database :operationalist nil :niladic t))
 (push '|Category| *allconstructors*)
 ; UNION
 (setf (get '|Union| 'database)
   (make-database :operationalist nil :constructorkind '|domain|))
 (push '|Union| *allconstructors*)
 ; RECORD
 (setf (get '|Record| 'database)
  (make-database :operationalist nil :constructorkind '|domain|))
 (push '|Record| *allconstructors*)
 ; MAPPING
 (setf (get '|Mapping| 'database)
  (make-database :operationalist nil :constructorkind '|domain|))
 (push '|Mapping| *allconstructors*)
 ; ENUMERATION
 (setf (get '|Enumeration| 'database)
  (make-database :operationalist nil :constructorkind '|domain|))
 (push '|Enumeration| *allconstructors*)
 )
```

```
  (final-name (root)
    (format nil "~a.daase~a" root ext))
  )
 (let (d)
  (declare (special |$constructorList| *sourcefiles* *compressvector*
                    *allconstructors* *operation-hash*))
  (do-symbols (symbol)
   (when (get symbol 'database)
    (setf (get symbol 'database) nil)))
  (setq *hascategory-hash* (make-hash-table :test #'equal))
  (setq *operation-hash* (make-hash-table))
  (setq *allconstructors* nil)
  (setq *compressvector* nil)
  (withSpecialConstructors)
  (localdatabase nil
     (list (list '|dir| (namestring (truename "./")) ))
     'make-database)
  (dolist (dir dirlist)
   (localdatabase nil
    (list (list '|dir| (namestring (truename (format nil "./~a" dir)))))
    'make-database))
;browse.daase
  (load (concatenate 'string (|getEnv| "AXIOM") "/autoload/topics"))  ;; hack
  (|browserAutoloadOnceTrigger|)
  (|mkTopicHashTable|)
  (setq |$constructorList| nil) ;; affects buildLibdb
  (setq *sourcefiles* (build-name-to-pamphlet-hash
    (concatenate 'string (|getEnv| "AXIOM")
      "/../../src/algebra/*.pamphlet")))
  (|buildLibdb|)
  (|dbSplitLibdb|)
; (|dbAugmentConstructorDataTable|)
  (|mkUsersHashTable|)
  (|saveUsersHashTable|)
  (|mkDependentsHashTable|)
  (|saveDependentsHashTable|)
; (|buildGloss|)
  (write-compress)
  (write-browsedb)
  (write-operationdb)
 ; note: genCategoryTable creates a new *hascategory-hash* table
 ; this smashes the existing table and regenerates it.
 ; write-categorydb does getdatabase calls to write the new information
  (write-categorydb)
  (dolist (con (|allConstructors|))
   (let (dbstruct)
     (when (setq dbstruct (get con 'database))
           (setf (database-cosig dbstruct)
                 (cons nil (mapcar #'|categoryForm?|
                           (cddar (database-constructormodemap dbstruct)))))
```

```
          (when (and (|categoryForm?| con)
                     (= (length (setq d (|domainsOf| (list con) NIL NIL))) 1))
                (setq d (caar d))
                (when (= (length d) (length (|getConstructorForm| con)))
              (format t "   ~a has a default domain of ~a~%" con (car d))
                     (setf (database-defaultdomain dbstruct) (car d)))))))
      ; note: genCategoryTable creates *ancestors-hash*. write-interpdb
      ; does gethash calls into it rather than doing a getdatabase call.
  (write-interpdb)
  (write-warmdata)
  (when (probe-file (final-name "compress"))
        (delete-file (final-name "compress")))
  (rename-file "compress.build" (final-name "compress"))
  (when (probe-file (final-name "interp"))
        (delete-file (final-name "interp")))
  (rename-file "interp.build" (final-name "interp"))
  (when (probe-file (final-name "operation"))
        (delete-file (final-name "operation")))
  (rename-file "operation.build" (final-name "operation"))
  (when (probe-file (final-name "browse"))
        (delete-file (final-name "browse")))
  (rename-file "browse.build"
               (final-name "browse"))
  (when (probe-file (final-name "category"))
        (delete-file (final-name "category")))
  (rename-file "category.build"
               (final-name "category")))))
```

———-

65.1.39 defun saveDependentsHashTable

[erase p??]
[writeLib1 p??]
[msort p??]
[hkeys p1038]
[rwrite p600]
[hget p1038]
[rshut p??]
[$depTb p??]
[$erase p??]

— **defun saveDependentsHashTable** —

```
(defun |saveDependentsHashTable| ()
 (let (stream)
 (declare (special |$depTb| $erase))
```

```
  ($erase '|dependents| 'database '|a|)
  (setq stream (|writeLib1| '|dependents| 'database '|a|))
  (dolist (k (msort (hkeys |$depTb|)))
    (|rwrite| k (hget |$depTb| k) stream))
  (rshut stream)))
```

———-

65.1.40 defun saveUsersHashTable

[erase p??]
[writeLib1 p??]
[msort p??]
[hkeys p1038]
[rwrite p600]
[hget p1038]
[rshut p??]
[$erase p??]
[$usersTb p??]

— **defun saveUsersHashTable** —

```
(defun |saveUsersHashTable| ()
 (let (stream)
 (declare (special |$usersTb| $erase))
  ($erase '|users| 'database '|a|)
  (setq stream (|writeLib1| '|users| 'database '|a|))
  (dolist (k (msort (hkeys |$usersTb|)))
    (|rwrite| k (HGET |$usersTb| k) stream))
  (rshut stream)))
```

———-

65.1.41 defun Construct the proper database full pathname

[getEnv p??]
[$spadroot p12]

— **defun DaaseName** —

```
(defun DaaseName (name erase?)
 (let (daase filename)
  (declare (special $spadroot))
  (if (setq daase (|getEnv| "DAASE"))
```

```
  (progn
   (setq filename  (concatenate 'string daase "/algebra/" name))
   (format t "   Using local database ~a.." filename))
  (setq filename (concatenate 'string $spadroot "/algebra/" name)))
 (when erase? (system::system (concatenate 'string "rm -f " filename)))
 filename))
```

———-

65.1.42 compress.daase

The compress database is special. It contains a list of symbols. The character string name of a symbol in the other databases is represented by a negative number. To get the real symbol back you take the absolute value of the number and use it as a byte index into the compress database. In this way long symbol names become short negative numbers.

65.1.43 defun Set up compression vectors for the databases

[DaaseName p1014]
[$spadroot p12]
[*compressvector* p988]
[*compressVectorLength* p988]
[*compress-stream* p989]
[*compress-stream-stamp* p989]

— **defun compressOpen** —

```
(defun compressOpen ()
 (let (lst stamp pos)
  (declare (special $spadroot *compressvector* *compressVectorLength*
                    *compress-stream* *compress-stream-stamp*))
  (setq *compress-stream*
    (open (DaaseName "compress.daase"  nil) :direction :input))
  (setq stamp (read *compress-stream*))
  (unless (equal stamp *compress-stream-stamp*)
   (format t "   Re-reading compress.daase")
   (setq *compress-stream-stamp* stamp)
   (setq pos (car stamp))
   (file-position *compress-stream* pos)
   (setq lst (read *compress-stream*))
   (setq *compressVectorLength* (car lst))
   (setq *compressvector*
     (make-array (car lst) :initial-contents (cdr lst))))))
```

———-

65.1.44 defvar $*attributes*

— **initvars** —

```
(defvar *attributes*
      '(|nil| |infinite| |arbitraryExponent| |approximate| |complex|
        |shallowMutable| |canonical| |noetherian| |central|
        |partiallyOrderedSet| |arbitraryPrecision| |canonicalsClosed|
        |noZeroDivisors| |rightUnitary| |leftUnitary|
        |additiveValuation| |unitsKnown| |canonicalUnitNormal|
        |multiplicativeValuation| |finiteAggregate| |shallowlyMutable|
        |commutative|) "The list of known algebra attributes")
```

————-

65.1.45 defun Write out the compress database

[allConstructors p1027]
[allOperations p1027]
[*compress-stream* p989]
[*attributes* p1016]
[*compressVectorLength* p988]

— **defun write-compress** —

```
(defun write-compress ()
 (let (compresslist masterpos out)
  (declare (special *compress-stream* *attributes* *compressVectorLength*))
  (close *compress-stream*)
  (setq out (open "compress.build" :direction :output))
  (princ "                              " out)
  (finish-output out)
  (setq masterpos (file-position out))
  (setq compresslist
        (append (|allConstructors|) (|allOperations|) *attributes*))
  (push "algebra" compresslist)
  (push "failed" compresslist)
  (push 'signature compresslist)
  (push '|ofType| compresslist)
  (push '|Join| compresslist)
  (push 'and compresslist)
  (push '|nobranch| compresslist)
  (push 'category compresslist)
  (push '|category| compresslist)
  (push '|domain| compresslist)
  (push '|package| compresslist)
```

```
  (push 'attribute compresslist)
  (push '|isDomain| compresslist)
  (push '|ofCategory| compresslist)
  (push '|Union| compresslist)
  (push '|Record| compresslist)
  (push '|Mapping| compresslist)
  (push '|Enumeration| compresslist)
  (setq *compressVectorLength* (length compresslist))
  (setq *compressvector*
    (make-array *compressVectorLength* :initial-contents compresslist))
  (print (cons (length compresslist) compresslist) out)
  (finish-output out)
  (file-position out 0)
  (print (cons masterpos (get-universal-time)) out)
  (finish-output out)
  (close out)))
```

———-

65.1.46 defun Compress an expression using the compress vector

This function is used to minimize the size of the databases by replacing symbols with indexes into the compression vector. [*compressvector* p988]

— **defun squeeze** —

```
(defun squeeze (expr)
 (declare (special *compressvector*))
 (let (leaves pos (bound (length *compressvector*)))
  (labels (
   (flat (expr)
    (when (and (numberp expr) (< expr 0) (>= expr bound))
     (print expr)
     (break "squeeze found a negative number"))
    (if (atom expr)
     (unless (or (null expr)
                 (and (symbolp expr) (char= (schar (symbol-name expr) 0) #\*)))
      (setq leaves (adjoin expr leaves)))
     (progn
      (flat (car expr))
      (flat (cdr expr))))))
  (setq leaves nil)
  (flat expr)
  (dolist (leaf leaves)
   (when (setq pos (position leaf *compressvector*))
     (nsubst (- pos) leaf expr)))
  expr)))
```

———-

65.1.47 defun Uncompress an expression using the compress vector

This function is used to recover symbols from the databases by using integers as indexes into the compression vector. [*compressvector* p988]

— **defun unsqueeze** —

```
(defun unsqueeze (expr)
 (declare (special *compressvector*))
  (cond ((atom expr)
         (cond ((and (numberp expr) (<= expr 0))
                (svref *compressVector* (- expr)))
               (t expr)))
        (t (rplaca expr (unsqueeze (car expr)))
           (rplacd expr (unsqueeze (cdr expr)))
           expr)))
```

———-

65.1.48 Building the interp.daase from hash tables

```
format of an entry in interp.daase:
 (constructor-name
   operationalist
   constructormodemap
   modemaps             -- this should not be needed. eliminate it.
   object               -- the name of the object file to load for this con.
   constructorcategory -- note that this info is the cadar of the
         constructormodemap for domains and packages so it is stored
         as NIL for them. it is valid for categories.
   niladic              -- t or nil directly
   unused
   cosig                -- kept directly
   constructorkind      -- kept directly
   defaultdomain        -- a short list, for %i
   ancestors            -- used to compute new category updates
 )
```

Here I'll try to outline the interp database write procedure

```
(defun write-interpdb ()
 "build interp.daase from hash tables"
 (declare (special $spadroot *ancestors-hash*))
 (let (opalistpos modemapspos cmodemappos master masterpos obj *print-pretty*
```

```
        concategory categorypos kind niladic cosig abbrev defaultdomain
        ancestors ancestorspos out)
  (declare (special *print-pretty*))
  (print "building interp.daase")

; 1. We open the file we're going to create

  (setq out (open "interp.build" :direction :output))

; 2. We reserve some space at the top of the file for the key-time pair
;    We will overwrite these spaces just before we close the file.

  (princ "                              " out)

; 3. Make sure we write it out
  (finish-output out)

; 4. For every constructor in the system we write the parts:

  (dolist (constructor (|allConstructors|))
   (let (struct)

; 4a. Each constructor has a property list. A property list is a list
;     of (key . value) pairs. The property we want is called 'database
;     so there is a ('database . something) in the property list

    (setq struct (get constructor 'database))

; 5 We write the "operationsalist"
; 5a. We remember the current file position before we write
;     We need this information so we can seek to this position on read

    (setq opalistpos (file-position out))

; 5b. We get the "operationalist", compress it, and write it out

    (print (squeeze (database-operationalist struct)) out)

; 5c. We make sure it was written

    (finish-output out)

; 6 We write the "constructormodemap"
; 6a. We remember the current file position before we write

    (setq cmodemappos (file-position out))

; 6b. We get the "constructormodemap", compress it, and write it out

    (print (squeeze (database-constructormodemap struct)) out)
```

```
; 6c. We make sure it was written

    (finish-output out)

; 7. We write the "modemaps"
; 7a. We remember the current file position before we write

    (setq modemapspos (file-position out))

; 7b. We get the "modemaps", compress it, and write it out

    (print (squeeze (database-modemaps struct)) out)

; 7c. We make sure it was written

    (finish-output out)

; 8. We remember source file pathnames in the obj variable

    (if (consp (database-object struct)) ; if asharp code ...
     (setq obj
      (cons (pathname-name (car (database-object struct)))
            (cdr (database-object struct))))
     (setq obj
      (pathname-name
        (first (last (pathname-directory (database-object struct)))))))

; 9. We write the "constructorcategory", if it is a category, else nil
; 9a. Get the constructorcategory and compress it

    (setq concategory (squeeze (database-constructorcategory struct)))

; 9b. If we have any data we write it out, else we don't write it
;     Note that if there is no data then the byte index for the
;     constructorcatagory will not be a number but will be nil.

    (if concategory  ; if category then write data else write nil
     (progn
      (setq categorypos (file-position out))
      (print concategory out)
      (finish-output out))
     (setq categorypos nil))

; 10. We get a set of properties which are kept as "immediate" data
;     This means that the key table will hold this data directly
;     rather than as a byte index into the file.
; 10a. niladic data

    (setq niladic (database-niladic struct))
```

```
; 10b. abbreviation data (e.g. POLY for polynomial)

    (setq abbrev (database-abbreviation struct))

; 10c. cosig data

    (setq cosig (database-cosig struct))

; 10d. kind data

    (setq kind (database-constructorkind struct))

; 10e. defaultdomain data

    (setq defaultdomain (database-defaultdomain struct))

; 11. The ancestor data might exist. If it does we fetch it,
;     compress it, and write it out. If it does not we place
;     and immediate value of nil in the key-value table

    (setq ancestors (squeeze (gethash constructor *ancestors-hash*))) ;cattable.boot
    (if ancestors
     (progn
      (setq ancestorspos (file-position out))
      (print ancestors out)
      (finish-output out))
     (setq ancestorspos nil))

; 12. "master" is an alist. Each element of the alist has the name of
;     the constructor and all of the above attributes. When the loop
;     finishes we will have constructed all of the data for the key-value
;     table

    (push (list constructor opalistpos cmodemappos modemapspos
      obj categorypos niladic abbrev cosig kind defaultdomain
      ancestorspos) master)))

; 13. The loop is done, we make sure all of the data is written

  (finish-output out)

; 14. We remember where the key-value table will be written in the file

  (setq masterpos (file-position out))

; 15. We compress and print the key-value table

  (print (mapcar #'squeeze master) out)
```

```
; 16. We make sure we write the table

  (finish-output out)

; 17. We go to the top of the file

  (file-position out 0)

; 18. We write out the (master-byte-position . universal-time) pair
;     Note that if the universal-time value matches the value of
;     *interp-stream-stamp* then there is no reason to read the
;     interp database because all of the data is already cached in
;     the image. This happens if you build a database and immediatly
;     save the image. The saved image already has the data since we
;     just wrote it out. If the *interp-stream-stamp* and the database
;     time stamp differ we "reread" the database on startup. Actually
;     we just open the database and fetch as needed. You can see fetches
;     by setting the *miss* variable non-nil.

  (print (cons masterpos (get-universal-time)) out)

; 19. We make sure we write it.

  (finish-output out)

; 20 And we are done

  (close out)))
```

65.1.49 defun Write the interp database

[squeeze p1017]
[$spadroot p12]
[*ancestors-hash* p**??**]
[*print-pretty* p**??**]

— defun write-interpdb —

```
(defun write-interpdb ()
 "build interp.daase from hash tables"
 (declare (special $spadroot *ancestors-hash*))
 (let (opalistpos modemapspos cmodemappos master masterpos obj *print-pretty*
       concategory categorypos kind niladic cosig abbrev defaultdomain
       ancestors ancestorspos out)
  (declare (special *print-pretty*))
  (print "building interp.daase")
  (setq out (open "interp.build" :direction :output))
  (princ "                              " out)
```

```
(finish-output out)
(dolist (constructor (|allConstructors|))
 (let (struct)
  (setq struct (get constructor 'database))
  (setq opalistpos (file-position out))
  (print (squeeze (database-operationalist struct)) out)
  (finish-output out)
  (setq cmodemappos (file-position out))
  (print (squeeze (database-constructormodemap struct)) out)
  (finish-output out)
  (setq modemapspos (file-position out))
  (print (squeeze (database-modemaps struct)) out)
  (finish-output out)
  (if (consp (database-object struct)) ; if asharp code ...
   (setq obj
    (cons (pathname-name (car (database-object struct)))
          (cdr (database-object struct))))
   (setq obj
    (pathname-name
      (first (last (pathname-directory (database-object struct)))))))
  (setq concategory (squeeze (database-constructorcategory struct)))
  (if concategory  ; if category then write data else write nil
   (progn
    (setq categorypos (file-position out))
    (print concategory out)
    (finish-output out))
   (setq categorypos nil))
  (setq niladic (database-niladic struct))
  (setq abbrev (database-abbreviation struct))
  (setq cosig (database-cosig struct))
  (setq kind (database-constructorkind struct))
  (setq defaultdomain (database-defaultdomain struct))
  (setq ancestors
   (squeeze (gethash constructor *ancestors-hash*))) ;cattable.boot
  (if ancestors
   (progn
    (setq ancestorspos (file-position out))
    (print ancestors out)
    (finish-output out))
   (setq ancestorspos nil))
  (push (list constructor opalistpos cmodemappos modemapspos
    obj categorypos niladic abbrev cosig kind defaultdomain
    ancestorspos) master)))
(finish-output out)
(setq masterpos (file-position out))
(print (mapcar #'squeeze master) out)
(finish-output out)
(file-position out 0)
(print (cons masterpos (get-universal-time)) out)
(finish-output out)
```

```
  (close out)))
```

———-

65.1.50 Building the browse.daase from hash tables

```
format of an entry in browse.daase:
( constructorname
    sourcefile
    constructorform
    documentation
    attributes
    predicates
)
```

This is essentially the same overall process as write-interpdb.

We reserve some space for the (key-table-byte-position . timestamp)

We loop across the list of constructors dumping the data and remembering the byte positions in a key-value pair table.

We dump the final key-value pair table, write the byte position and time stamp at the top of the file and close the file.

65.1.51 defun Write the browse database

[allConstructors p1027]
[squeeze p1017]
[$spadroot p12]
[*sourcefiles* p**??**]
[*print-pretty* p**??**]

— **defun write-browsedb** —

```
(defun write-browsedb ()
 "make browse.daase from hash tables"
 (declare (special $spadroot *sourcefiles*))
 (let (master masterpos src formpos docpos attpos predpos *print-pretty* out)
  (declare (special *print-pretty*))
  (print "building browse.daase")
  (setq out (open "browse.build" :direction :output))
  (princ "                              " out)
  (finish-output out)
  (dolist (constructor (|allConstructors|))
   (let (struct)
    (setq struct (get constructor 'database))
```

```
    ; sourcefile is small. store the string directly
   (setq src (gethash constructor *sourcefiles*))
   (setq formpos (file-position out))
   (print (squeeze (database-constructorform struct)) out)
   (finish-output out)
   (setq docpos (file-position out))
   (print (database-documentation struct) out)
   (finish-output out)
   (setq attpos (file-position out))
   (print (squeeze (database-attributes struct)) out)
   (finish-output out)
   (setq predpos (file-position out))
   (print (squeeze (database-predicates struct)) out)
   (finish-output out)
   (push (list constructor src formpos docpos attpos predpos) master)))
 (finish-output out)
 (setq masterpos (file-position out))
 (print (mapcar #'squeeze master) out)
 (finish-output out)
 (file-position out 0)
 (print (cons masterpos (get-universal-time)) out)
 (finish-output out)
 (close out)))
```

———-

65.1.52 Building the category.daase from hash tables

This is a single table of category hash table information, dumped in the database format.

65.1.53 defun Write the category database

[genCategoryTable p**??**]
[squeeze p1017]
[*print-pretty* p**??**]
[*hasCategory-hash* p987]

— **defun write-categorydb** —

```
(defun write-categorydb ()
 "make category.daase from scratch. contains the *hasCategory-hash* table"
 (let (out master pos *print-pretty*)
  (declare (special *print-pretty* *hasCategory-hash*))
  (print "building category.daase")
  (|genCategoryTable|)
  (setq out (open "category.build" :direction :output))
```

```
  (princ "                              " out)
  (finish-output out)
  (maphash #'(lambda (key value)
    (if (or (null value) (eq value t))
     (setq pos value)
     (progn
      (setq pos (file-position out))
      (print (squeeze value) out)
      (finish-output out)))
     (push (list key pos) master))
     *hasCategory-hash*)
  (setq pos (file-position out))
  (print (mapcar #'squeeze master) out)
  (finish-output out)
  (file-position out 0)
  (print (cons pos (get-universal-time)) out)
  (finish-output out)
  (close out)))
```

———-

65.1.54 Building the operation.daase from hash tables

This is a single table of operations hash table information, dumped in the database format.

65.1.55 defun Write the operations database

[squeeze p1017]
[*operation-hash* p987]

— defun write-operationdb —

```
(defun write-operationdb ()
 (let (pos master out)
  (declare (special leaves *operation-hash*))
  (setq out (open "operation.build" :direction :output))
  (princ "                              " out)
  (finish-output out)
  (maphash #'(lambda (key value)
   (setq pos (file-position out))
   (print (squeeze value) out)
   (finish-output out)
   (push (cons key pos) master))
   *operation-hash*)
  (finish-output out)
  (setq pos (file-position out))
```

```
  (print (mapcar #'squeeze master) out)
  (file-position out 0)
  (print (cons pos (get-universal-time)) out)
  (finish-output out)
  (close out)))
```

———-

65.1.56 Database support operations

65.1.57 defun Data preloaded into the image at build time

[$topicHash p??]

— **defun write-warmdata** —

```
(defun write-warmdata ()
 "write out information to be loaded into the image at build time"
 (declare (special |$topicHash|))
 (with-open-file (out "warm.data" :direction :output)
  (format out "(in-package \"BOOT\")~%")
  (format out "(setq |$topicHash| (make-hash-table))~%")
  (maphash #'(lambda (k v)
   (format out "(setf (gethash '|~a| |$topicHash|) ~a)~%" k v)) |$topicHash|)))
```

———-

65.1.58 defun Return all constructors

[*allconstructors* p991]

— **defun allConstructors** —

```
(defun |allConstructors| ()
 (declare (special *allconstructors*))
 *allconstructors*)
```

———-

65.1.59 defun Return all operations

[*allOperations* p991]
[*operation-hash* p987]

— defun allOperations —

```
(defun |allOperations| ()
 (declare (special *allOperations* *operation-hash*))
 (unless *allOperations*
  (maphash #'(lambda (k v) (declare (ignore v)) (push k *allOperations*))
    *operation-hash*))
 *allOperations*)
```

———-

Chapter 66

System Statistics

[gbc-time p??]

— **defun statisticsInitialization** —

```
(defun |statisticsInitialization| ()
 "initialize the garbage collection timer"
 #+:akcl (system:gbc-time 0)
 nil)
```

———-

66.1 Lisp Library Handling

66.1.1 defun loadLib

[startTimingProcess p??]
[getdatabase p1001]
[isSystemDirectory p1030]
[pathnameDirectory p1035]
[loadLibNoUpdate p1031]
[sayKeyedMsg p327]
[namestring p1034]
[clearConstructorCache p??]
[updateDatabase p1009]
[installConstructor p??]
[updateCategoryTable p??]
[categoryForm? p??]
[remprop p??]

[stopTimingProcess p**??**]
[$InteractiveMode p24]
[$printLoadMsgs p727]
[$forceDatabaseUpdate p**??**]
[$CategoryFrame p**??**]

— defun loadLib —

```
(defun |loadLib| (cname)
 (let (fullLibName systemdir? update? kind u sig coSig)
 (declare (special |$CategoryFrame| |$InteractiveMode| |$printLoadMsgs|
                   |$forceDatabaseUpdate|))
  (|startTimingProcess| '|load|)
  (when (setq fullLibName (getdatabase cname 'object))
   (setq systemdir? (|isSystemDirectory| (|pathnameDirectory| fullLibName)))
   (setq update? (or |$forceDatabaseUpdate| (null systemdir?)))
   (cond
    ((null update?) (|loadLibNoUpdate| cname cname fullLibName))
    (t
     (setq kind (getdatabase cname 'constructorkind))
     (when |$printLoadMsgs|
       (|sayKeyedMsg| 'S2IL0002 (list (|namestring| fullLibName) kind cname)))
     (load fullLibName)
     (|clearConstructorCache| cname)
     (|updateDatabase| cname cname systemdir?)
     (|installConstructor| cname kind)
     (setq u (getdatabase cname 'constructormodemap))
     (|updateCategoryTable| cname kind)
     (setq coSig
      (when u
        (setq sig (cdar u))
        (cons nil (loop for x in (cdr sig) collect (|categoryForm?| x)))))
     (if (null (cdr (getdatabase cname 'constructorform)))
       (setf (get cname 'niladic) t)
       (remprop cname 'niladic))
     (setf (get cname 'loaded) fullLibName)
     (when |$InteractiveMode| (setq |$CategoryFrame| (list (list nil))))
     (|stopTimingProcess| '|load|)
     t)))))
```

———-

66.1.2 defun isSystemDirectory

[function p**??**]
[$spadroot p12]

— **defun isSystemDirectory** —

```
(defun |isSystemDirectory| (dir)
 (declare (special $spadroot))
 (every (|function| char=) $spadroot dir))
```

———-

66.1.3 defun loadLibNoUpdate

[getdatabase p1001]
[sayKeyedMsg p327]
[toplevel p??]
[clearConstructorCache p??]
[installConstructor p??]
[stopTimingProcess p??]
[$printLoadMsgs p727]
[$InteractiveMode p24]
[$CategoryFrame p??]

— **defun loadLibNoUpdate** —

```
(defun |loadLibNoUpdate| (cname libName fullLibName)
 (declare (ignore libName))
 (let (kind)
 (declare (special |$CategoryFrame| |$InteractiveMode| |$printLoadMsgs|))
  (setq kind (getdatabase cname 'constructorkind))
  (when |$printLoadMsgs|
    (|sayKeyedMsg| 'S2IL0002 (list (|namestring| fullLibName) kind cname)))
  (cond
   ((equal (catch 'versioncheck (load fullLibName)) (- 1))
     (princ "   wrong library version...recompile ")
     (princ fullLibName)
     (terpri)
     (toplevel))
   (t
     (|clearConstructorCache| cname)
     (|installConstructor| cname kind)
     (setf (get cname 'loaded) fullLibName)
     (when |$InteractiveMode| (setq |$CategoryFrame| (list (list nil))))
     (|stopTimingProcess| '|load|)))
  t))
```

———-

66.1.4 defun loadFunctor

[loadFunctor p1032]
[loadLibIfNotLoaded p??]

— **defun loadFunctor** —

```
(defun |loadFunctor| (u)
 (cond
   ((null (atom u)) (|loadFunctor| (car u)))
   (t
     (|loadLibIfNotLoaded| u)
     u)))
```

————-

Chapter 67

Special Lisp Functions

67.1 Axiom control structure macros

Axiom used various control structures in the boot code which are not available in Common Lisp. We write some macros here to make the boot to lisp translations easier to read.

67.1.1 defun put

— **defun put** —

```
(defun put (sym ind val) (setf (get sym ind) val))
```

————-

67.1.2 defmacro while

While the condition is true, repeat the body. When the condition is false, return t.

— **defmacro while** —

```
(defmacro while (condition &rest body)
 `(loop (if (not ,condition) (return t)) ,@body))
```

————-

67.1.3 defmacro whileWithResult

While the condition is true, repeat the body. When the condition is false, return the result form's value.

— **defmacro whileWithResult** —

```
(defmacro whileWithResult (condition result &rest body)
 `(loop (if (not ,condition) ,@result) ,@body))
```

———-

67.2 Filename Handling

This code implements the Common Lisp pathname functions for Lisp/VM. On VM, a filename is 3-list consisting of the filename, filetype and filemode. We also UPCASE everything.

67.2.1 defun namestring

[pathname p1036]

— **defun namestring** —

```
(defun |namestring| (arg)
 (namestring (|pathname| arg)))
```

———-

67.2.2 defun pathnameName

[pathname p1036]

— **defun pathnameName** —

```
(defun |pathnameName| (arg)
 (pathname-name (|pathname| arg)))
```

———-

67.2.3 defun pathnameType

[pathname p1036]

— **defun pathnameType** —

```
(defun |pathnameType| (arg)
 (pathname-type (|pathname| arg)))
```

————-

67.2.4 defun pathnameTypeId

[upcase p**??**]
[object2Identifier p**??**]
[pathnameType p1034]

— **defun pathnameTypeId** —

```
(defun |pathnameTypeId| (arg)
 (upcase (|object2Identifier| (|pathnameType| arg))))
```

————-

67.2.5 defun mergePathnames

[pathnameName p1034]
[pathnameType p1034]
[pathnameDirectory p1035]

— **defun mergePathnames** —

```
(defun |mergePathnames| (a b)
 (let (fn ft fm)
  (cond
   ((string= (setq fn (|pathnameName| a)) "*") b)
   ((not (equal fn (|pathnameName| b))) a)
   ((string= (setq ft (|pathnameType| a)) "*") b)
   ((not (equal ft (|pathnameType| b))) a)
   ((equal (setq fm (|pathnameDirectory| a)) (list "*" )) b)
   (t a))))
```

————-

67.2.6 defun pathnameDirectory

[pathname p1036]

— **defun pathnameDirectory** —

```
(defun |pathnameDirectory| (arg)
 (namestring (make-pathname :directory (pathname-directory (|pathname| arg)))))
```

———-

67.2.7 defun Axiom pathnames

[pathname p1036]
[make-filename p**??**]

— **defun pathname** —

```
(defun |pathname| (p)
 (cond
  ((null p) p)
  ((pathnamep p) p)
  ((null (consp p)) (pathname p))
  (t
   (when (> (|#| p) 2) (setq p (cons (elt p 0) (cons (elt p 1) nil))))
   (pathname (apply #'make-filename p)))))
```

———-

67.2.8 defun makePathname

[pathname p1036]
[object2String p**??**]

— **defun makePathname** —

```
(defun |makePathname| (name type dir)
 (declare (ignore dir))
 (|pathname| (list (|object2String| name) (|object2String| type))))
```

———-

67.2.9 defun Delete a file

[erase p**??**]
[pathname p1036]

[$erase p**??**]

— **defun deleteFile** —

```
(defun |deleteFile| (arg)
 (declare (special $erase))
 ($erase (|pathname| arg)))
```

————-

67.2.10 defun wrap

[lotsof p1037]
[wrap p1037]

— **defun wrap** —

```
(defun wrap (list-of-items wrapper)
 (prog nil
  (cond
   ((or (not (consp list-of-items)) (not wrapper))
    (return list-of-items))
   ((not (consp wrapper))
    (setq wrapper (lotsof wrapper))))
  (return
    (cons
     (if (first wrapper)
      `(,(first wrapper) ,(first list-of-items))
       (first list-of-items))
     (wrap (cdr list-of-items) (cdr wrapper))))))
```

————-

67.2.11 defun lotsof

— **defun lotsof** —

```
(defun lotsof (&rest items)
  (setq items (copy-list items))
  (nconc items items))
```

————-

67.2.12 defmacro startsId?

— defmacro startsId? —

```
(defmacro |startsId?| (x)
 `(or (alpha-char-p ,x) (member ,x '(#\? #\% #\!) :test #'char=)))
```

————-

67.2.13 defun hput

— defun hput —

```
(defun hput (table key value)
 (setf (gethash key table) value))
```

————-

67.2.14 defmacro hget

— defmacro hget —

```
(defmacro HGET (table key &rest default)
   `(gethash ,key ,table ,@default))
```

————-

67.2.15 defun hkeys

— defun hkeys —

```
(defun hkeys (table)
 (let (keys)
  (maphash
   #'(lambda (key val) (declare (ignore val)) (push key keys)) table)
   keys))
```

————-

67.2.16 defun digitp

[digitp p1039]

— **defun digitp** —

```
(defun digitp (x)
  (or (and (symbolp x) (digitp (symbol-name x)))
      (and (characterp x) (digit-char-p x))
      (and (stringp x) (= (length x) 1) (digit-char-p (char x 0)))))
```

————-

67.2.17 defun pname

Note it is important that PNAME returns nil not an error for non-symbols

— **defun pname 0** —

```
(defun pname (x)
  (cond ((symbolp x) (symbol-name x))
        ((characterp x) (string x))
        (t nil)))
```

————-

67.2.18 defun size

— **defun size** —

```
(defun size (l)
 (cond
  ((vectorp l) (length l))
  ((consp l)   (list-length l))
  (t 0)))
```

————-

67.2.19 defun strpos

— **defun strpos** —

```
(defun strpos (what in start dontcare)
 (setq what (string what) in (string in))
 (if dontcare
  (progn
   (setq dontcare (character dontcare))
   (search what in :start2 start
                   :test #'(lambda (x y) (or (eql x dontcare) (eql x y)))))
  (if (= start 0)
   (search what in)
   (search what in :start2 start))))
```

———-

67.2.20 defun strposl

Note that this assumes "table" is a string.

— **defun strposl** —

```
(defun strposl (table cvec sint item)
 (setq cvec (string cvec))
 (if (not item)
  (position table cvec :test #'(lambda (x y) (position y x)) :start sint)
  (position table cvec :test-not #'(lambda (x y) (position y x)) :start sint)))
```

———-

67.2.21 defun qenum

— **defun qenum 0** —

```
(defun qenum (cvec ind)
 (char-code (char cvec ind)))
```

———-

67.2.22 defmacro identp

— **defmacro identp 0** —

```
(defmacro identp (x)
 (if (atom x)
```

```
  `(and ,x (symbolp ,x))
   (let ((xx (gensym)))
    `(let ((,xx ,x))
      (and ,xx (symbolp ,xx))))))
```

————-

67.2.23 defun concat

[string-concatenate p**??**]

— **defun concat 0** —

```
(defun concat (a b &rest l)
 (if (bit-vector-p a)
   (if l
     (apply #'concatenate 'bit-vector a b l)
     (concatenate 'bit-vector a b))
   (if l
     (apply #'system:string-concatenate a b l)
     (system:string-concatenate a b))))
```

————-

67.2.24 defun functionp

[identp p1040]

— **defun functionp** —

```
(defun |functionp| (fn)
 (if (identp fn)
  (and (fboundp fn) (not (macro-function fn)))
  (functionp fn)))
```

————-

;; ————————¿ NEW DEFINITION (override in msgdb.boot.pamphlet)

67.2.25 defun brightprint

[messageprint p1042]

— **defun brightprint** —

```
(defun brightprint (x)
 (messageprint x))
```

————-

;; ————————¿ NEW DEFINITION (override in msgdb.boot.pamphlet)

67.2.26 defun brightprint-0

[messageprint-1 p1043]

— **defun brightprint-0** —

```
(defun brightprint-0 (x)
 (messageprint-1 x))
```

————-

67.2.27 defun member

— **defun member 0** —

```
(defun |member| (item sequence)
 (cond
   ((symbolp item) (member item sequence :test #'eq))
   ((stringp item) (member item sequence :test #'equal))
   ((and (atom item) (not (arrayp item))) (member item sequence))
   (t (member item sequence :test #'equalp))))
```

————-

67.2.28 defun messageprint

— **defun messageprint** —

```
(defun messageprint (x)
 (mapc #'messageprint-1 x))
```

————-

67.2.29 defun messageprint-1

[identp p1040]
[messageprint-1 p1043]
[messageprint-2 p1043]

— **defun messageprint-1** —

```
(defun messageprint-1 (x)
 (cond
  ((or (eq x '|%l|) (equal x "%l")) (terpri))
  ((stringp x) (princ x))
  ((identp x) (princ x))
  ((atom x) (princ x))
  ((princ "(")
   (messageprint-1 (car x))
   (messageprint-2 (cdr x))
   (princ ")"))))
```

———-

67.2.30 defun messageprint-2

[messageprint-1 p1043]
[messageprint-2 p1043]

— **defun messageprint-2** —

```
(defun messageprint-2 (x)
  (if (atom x)
   (unless x (progn (princ " . ") (messageprint-1 x)))
   (progn (princ " ") (messageprint-1 (car x)) (messageprint-2 (cdr x)))))
```

———-

67.2.31 defun sayBrightly1

[brightprint-0 p1042]
[brightprint p1041]

— **defun sayBrightly1** —

```
(defun sayBrightly1 (x *standard-output*)
  (if (atom x)
```

```
(progn (brightprint-0 x) (terpri) (force-output))
(progn (brightprint   x) (terpri) (force-output))))
```

67.2.32 defmacro assq

TPDHERE: This could probably be replaced by the default assoc using eql
— defmacro assq —

```
(defmacro assq (a b)
 `(assoc ,a ,b :test #'eq))
```

Chapter 68

Record, Union, Mapping, and Enumeration

— **postvars** —

```
(eval-when (eval load)
 (mapcar #'(lambda (alist)
             (setf (get (first alist) '|makeFunctionList|) (second alist)))
 '((|Record| |mkRecordFunList|)
   (|Union| |mkUnionFunList|)
   (|Mapping| |mkMappingFunList|)
   (|Enumeration| |mkEnumerationFunList|))))
```

———-

Chapter 69

Common Lisp Algebra Support

These functions are called directly from the algebra source code. They fall into two basic categories, one are the functions that are raw Comon Lisp calls and the other are Axiom specific functions or macros.

Raw function calls are used where there is an alignment of the Axiom type and the underlying representation in Common Lisp. These form the support pillars upon which Axiom rests. For instance, the 'EQ' function is called to support the Axiom equivalent 'eq?' function.

Macros are used to add type information in order to make low level operations faster. An example is the use of macros in DoubleFloat to add Common Lisp type information. Since DoubleFloat is machine arithmetic we give the compiler explicit type information so it can generate fast code.

Functions are used to do manipulations which are Common Lisp operations but the Axiom semantics are not the same. Because Axiom was originally written in Maclisp, then VMLisp, and then Common Lisp some of these old semantics survive.

69.1 Void

69.1.1 defun voidValue

— **defun voidValue** —

```
(defun |voidValue| () "()")
```

———-

69.2 U32Vector

69.2.1 defun getrefv32

— **defun getrefv32** —

```
(defun getrefv32 (n x)
 (make-array n :initial-element x :element-type '(unsigned-byte 32)))
```

———-

69.2.2 defmacro qv32len

— **defmacro qv32len** —

```
(defmacro qv32len (v)
 `(length (the (simple-array (unsigned-byte 32) (*)) ,v)))
```

———-

69.2.3 defmacro elt32

— **defmacro elt32** —

```
(defmacro elt32 (v i)
 `(aref (the (simple-array (unsigned-byte 32) (*)) ,v) ,i))
```

———-

69.2.4 defmacro setelt32

— **defmacro setelt32** —

```
(defmacro setelt32 (v i s)
 `(setf (aref (the (simple-array (unsigned-byte 32) (*)) ,v) ,i), s))
```

———-

69.3 DirectProduct

69.3.1 defun vec2list

— **defun vec2list** —

```
(defun vec2list (vec) (coerce vec 'list))
```

———-

69.4 AlgebraicFunction

69.4.1 defun retract

[objMode p??]
[objVal p??]
[isWrapped p??]
[qcar p??]
[retract1 p??]
[objNew p??]
[$EmptyMode p??]

— **defun retract** —

```
(defun |retract| (object)
 (labels (
  (retract1 (object)
   (let (type val underDomain objectp)
   (declare (special |$SingleInteger| |$Integer| |$NonNegativeInteger|
                     |$PositiveInteger|))
    (setq type (|objMode| object))
    (cond
     ((stringp type) '|failed|)
     (t
       (setq val (|objVal| object))
       (cond
        ((equal type |$PositiveInteger|) (|objNew| val |$NonNegativeInteger|))
        ((equal type |$NonNegativeInteger|) (|objNew| val |$Integer|))
        ((and (equal type |$Integer|) (typep (|unwrap| val) 'fixnum))
          (|objNew| val |$SingleInteger|))
        (t
          (cond
           ((or (eql 1 (|#| type))
                (and (consp type) (eq (qcar type) '|Union|))
```

```
              (and (consp type) (eq (qcar type) '|FunctionCalled|)
                   (and (consp (qcdr type)) (eq (qcddr type) nil)))
              (and (consp type) (eq (qcar type) '|OrderedVariableList|)
                   (and (consp (qcdr type)) (eq (qcddr type) nil)))
              (and (consp type) (eq (qcar type) '|Variable|)
                   (and (consp (qcdr type)) (eq (qcddr type) nil))))
            (if (setq objectp (|retract2Specialization| object))
              objectp
              '|failed|))
           ((null (setq underDomain (|underDomainOf| type)))
             '|failed|)
          ; try to retract the "coefficients", e.g. P RN -> P I or M RN -> M I
           (t
            (setq objectp (|retractUnderDomain| object type underDomain))
            (cond
             ((not (eq objectp '|failed|)) objectp)
             ; see if we can use the retract functions
             ((setq objectp (|coerceRetract| object underDomain)) objectp)
             ; see if we have a special case here
             ((setq objectp (|retract2Specialization| object)) objectp)
            (t '|failed|))))))))))
(let (type val ans)
(declare (special |$EmptyMode|))
 (setq type (|objMode| object))
 (cond
  ((stringp type) '|failed|)
  ((equal type |$EmptyMode|) '|failed|)
  (t
   (setq val (|objVal| object))
   (cond
    ((and (null (|isWrapped| val))
          (null (and (consp val) (eq (qcar val) 'map))))
      '|failed|)
    (t
     (cond
      ((eq (setq ans (retract1 (|objNew| val type))) '|failed|)
        ans)
      (t
       (|objNew| (|objVal| ans) (|objMode| ans)))))))))))
```

———-

69.5 Any

69.5.1 defun spad2BootCoerce

— defun spad2BootCoerce —

```
(defun |spad2BootCoerce| (x source target)
 (let (xp)
  (cond
   ((null (|isValidType| source)) (|throwKeyedMsg| 'S2IE0004 (list source)))
   ((null (|isValidType| target)) (|throwKeyedMsg| 'S2IE0004 (list target)))
   ((setq xp (|coerceInteractive| (|objNewWrap| x source) target))
     (|objValUnwrap| xp))
   (t
    (|throwKeyedMsgCannotCoerceWithValue| (|wrap| x) source target)))))
```

———-

69.6 ParametricLinearEquations

69.6.1 defun algCoerceInteractive

— defun algCoerceInteractive —

```
(defun |algCoerceInteractive| (p source target)
 (let (|$useConvertForCoercions| u)
 (declare (special |$useConvertForCoercions|))
  (setq |$useConvertForCoercions| t)
  (setq source (|devaluate| source))
  (setq target (|devaluate| target))
  (setq u (|coerceInteractive| (|objNewWrap| p source) target))
  (if u
   (|objValUnwrap| u)
   (|error| (list "can't convert" p "of mode" source "to mode" target)))))
```

———-

69.7 NumberFormats

69.7.1 defun ncParseFromString

— defun ncParseFromString —

```
(defun |ncParseFromString| (s)
  (|zeroOneTran| (catch 'SPAD_READER (|parseFromString| s))))
```

———-

69.8 SingleInteger

69.8.1 defun qsquotient

— defun qsquotient 0 —

```
(defun qsquotient (a b)
 (the fixnum (truncate (the fixnum a) (the fixnum b))))
```

———-

69.8.2 defun qsremainder

— defun qsremainder 0 —

```
(defun qsremainder (a b)
 (the fixnum (rem (the fixnum a) (the fixnum b))))
```

———-

69.8.3 defmacro qsdifference

— defmacro qsdifference 0 —

```
(defmacro qsdifference (x y)
 `(the fixnum (- (the fixnum ,x) (the fixnum ,y))))
```

———-

69.8.4 defmacro qslessp

— defmacro qslessp 0 —

```
(defmacro qslessp (a b)
 `(< (the fixnum ,a) (the fixnum ,b)))
```

———-

69.8.5 defmacro qsadd1

— defmacro qsadd1 0 —

```
(defmacro qsadd1 (x)
 `(the fixnum (1+ (the fixnum ,x))))
```

———-

69.8.6 defmacro qssub1

— defmacro qssub1 0 —

```
(defmacro qssub1 (x)
 `(the fixnum (1- (the fixnum ,x))))
```

———-

69.8.7 defmacro qsminus

— defmacro qsminus 0 —

```
(defmacro qsminus (x)
 `(the fixnum (minus (the fixnum ,x))))
```

———-

69.8.8 defmacro qsplus

— **defmacro qsplus 0** —

```
(defmacro qsplus (x y)
 `(the fixnum (+ (the fixnum ,x) (the fixnum ,y))))
```

———-

69.8.9 defmacro qstimes

— **defmacro qstimes 0** —

```
(defmacro qstimes (x y)
 `(the fixnum (* (the fixnum ,x) (the fixnum ,y))))
```

———-

69.8.10 defmacro qsabsval

— **defmacro qsabsval 0** —

```
(defmacro qsabsval (x)
  `(the fixnum (abs (the fixnum ,x))))
```

———-

69.8.11 defmacro qsoddp

— **defmacro qsoddp 0** —

```
(defmacro qsoddp (x)
 `(oddp (the fixnum ,x)))
```

———-

69.8.12 defmacro qszerop

— defmacro qszerop 0 —

```
(defmacro qszerop (x)
 `(zerop (the fixnum ,x)))
```

————-

69.8.13 defmacro qsmax

— defmacro qsmax 0 —

```
(defmacro qsmax (x y)
 `(the fixnum (max (the fixnum ,x) (the fixnum ,y))))
```

————-

69.8.14 defmacro qsmin

— defmacro qsmin 0 —

```
(defmacro qsmin (x y)
 `(the fixnum (min (the fixnum ,x) (the fixnum ,y))))
```

————-

69.9 Boolean

69.9.1 defun The Boolean = function support

— defun BooleanEquality 0 —

```
(defun |BooleanEquality| (x y) (if x y (null y)))
```

————-

69.10 IndexedBits

69.10.1 defmacro truth-to-bit

IndexedBits new function support
— **defmacro truth-to-bit** —

```
(defmacro truth-to-bit (x) `(cond (,x 1) ('else 0)))
```

———-

69.10.2 defun IndexedBits new function support

— **defun bvec-make-full 0** —

```
(defun bvec-make-full (n x)
 (make-array (list n) :element-type 'bit :initial-element x))
```

———-

69.10.3 defmacro bit-to-truth

IndexedBits elt function support
— **defmacro bit-to-truth 0** —

```
(defmacro bit-to-truth (b) `(eq ,b 1))
```

———-

69.10.4 defmacro bvec-elt

IndexedBits elt function support
— **defmacro bvec-elt 0** —

```
(defmacro bvec-elt (bv i) `(sbit ,bv ,i))
```

———-

69.10.5 defmacro bvec-setelt

IndexedBits setelt function support
— **defmacro bvec-setelt** —

```
(defmacro bvec-setelt (bv i x) `(setf (sbit ,bv ,i) ,x))
```

———-

69.10.6 defmacro bvec-size

IndexedBits length function support
— **defmacro bvec-size** —

```
(defmacro bvec-size (bv) `(size ,bv))
```

———-

69.10.7 defun IndexedBits concat function support

— **defun bvec-concat 0** —

```
(defun bvec-concat (bv1 bv2) (concatenate '(vector bit) bv1 bv2))
```

———-

69.10.8 defun IndexedBits copy function support

— **defun bvec-copy 0** —

```
(defun bvec-copy (bv) (copy-seq bv))
```

———-

69.10.9 defun IndexedBits = function support

— **defun bvec-equal 0** —

```
(defun bvec-equal (bv1 bv2) (equal bv1 bv2))
```

———-

69.10.10 defun IndexedBits < function support

— defun bvec-greater 0 —

```
(defun bvec-greater (bv1 bv2)
  (let ((pos (mismatch bv1 bv2)))
    (cond ((or (null pos) (>= pos (length bv1))) nil)
          ((< pos (length bv2)) (> (bit bv1 pos) (bit bv2 pos)))
          ((find 1 bv1 :start pos) t)
          (t nil))))
```

———-

69.10.11 defun IndexedBits And function support

— defun bvec-and 0 —

```
(defun bvec-and (bv1 bv2) (bit-and  bv1 bv2))
```

———-

69.10.12 defun IndexedBits Or function support

— defun bvec-or 0 —

```
(defun bvec-or (bv1 bv2) (bit-ior  bv1 bv2))
```

———-

69.10.13 defun IndexedBits xor function support

— defun bvec-xor 0 —

```
(defun bvec-xor (bv1 bv2) (bit-xor  bv1 bv2))
```

———-

69.10.14 defun IndexedBits nand function support

— **defun bvec-nand 0** —

```
(defun bvec-nand (bv1 bv2) (bit-nand bv1 bv2))
```

———-

69.10.15 defun IndexedBits nor function support

— **defun bvec-nor 0** —

```
(defun bvec-nor (bv1 bv2) (bit-nor  bv1 bv2))
```

———-

69.10.16 defun IndexedBits not function support

— **defun bvec-not 0** —

```
(defun bvec-not (bv) (bit-not  bv))
```

———-

69.11 KeyedAccessFile

69.11.1 defun KeyedAccessFile defstream function support

This is a simpler interpface to RDEFIOSTREAM [rdefiostream p**??**]

— **defun rdefinstream** —

```
(defun rdefinstream (&rest fn)
  ;; following line prevents rdefiostream from adding a default filetype
  (unless (rest fn) (setq fn (list (pathname (car fn)))))
  (rdefiostream (list (cons 'file fn) '(mode . input))))
```

———-

69.11.2 defun KeyedAccessFile defstream function support

[rdefiostream p**??**]

— **defun rdefoutstream** —

```
(defun rdefoutstream (&rest fn)
  ;; following line prevents rdefiostream from adding a default filetype
  (unless (rest fn) (setq fn (list (pathname (car fn)))))
  (rdefiostream (list (cons 'FILE fn) '(mode . OUTPUT))))
```

———-

69.12 Table

69.12.1 defun Table InnerTable support

We look inside the Key domain given to Table and find if there is an equality predicate associated with the domain. If found then Table will use a HashTable representation, otherwise it will use an AssociationList representation [knownEqualPred p**??**]
[compiledLookup p1061]
[Boolean p**??**]
[bpiname p**??**]
[knownEqualPred p**??**]

— **defun hashable** —

```
(defun |hashable| (dom)
 (labels (
  (|knownEqualPred| (dom)
    (let ((fun (|compiledLookup| '= '((|Boolean|) $ $) dom)))
     (if fun
      (get (bpiname (car fun)) '|SPADreplace|)
      nil))))
  (member (|knownEqualPred| dom) '(eq eql equal))))
```

———-

69.12.2 defun compiledLookup

[isDomain p??]
[NRTevalDomain p1064]

— **defun compiledLookup** —

```
(defun |compiledLookup| (op sig dollar)
 (unless (|isDomain| dollar) (setq dollar (|NRTevalDomain| dollar)))
 (|basicLookup| op sig dollar dollar))
```

———-

69.12.3 defun basicLookup

[spadcall p??]
[hashCode? p??]
[opIsHasCat p??]
[HasCategory p??]
[hashType p??]
[hashString p??]
[error p??]
[vecp p??]
[isNewWorldDomain p??]
[oldCompLookup p1064]
[lookupInDomainVector p1063]
[$hashSeg p??]
[$hashOpSet p??]
[$hashOpApply p??]
[$hashOp0 p??]
[$hashOp1 p??]

— **defun basicLookup** —

```
(defun |basicLookup| (op sig domain dollar)
 (let (hashPercent box dispatch lookupFun hashSig val boxval)
 (declare (special |$hashSeg| |$hashOpSet| |$hashOpApply| |$hashOp0|
                   |$hashOp1|))
  (cond
   ((vecp domain)
    (if (|isNewWorldDomain| domain)
      (|oldCompLookup| op sig domain dollar)
      (|lookupInDomainVector| op sig domain dollar)))
  (t
   (setq hashPercent
    (if (vecp dollar)
```

```
   (|hashType| (elt dollar 0) 0)
   (|hashType| dollar 0)))
 (setq box (cons nil nil))
 (cond
  ((null (vecp (setq dispatch (car domain))))
    (|error| '|bad domain format|))
  (t
    (setq lookupFun (elt dispatch 3))
    (cond
      ((eql (elt dispatch 0) 0)
        (setq hashSig
         (cond
          ((|hashCode?| sig) sig)
          ((|opIsHasCat| op) (|hashType| sig hashPercent))
          (t (|hashType| (cons '|Mapping| sig) hashPercent))))
        (when (symbolp op)
         (cond
           ((eq op '|Zero|)   (setq op |$hashOp0|))
           ((eq op '|One|)    (setq op |$hashOp1|))
           ((eq op '|elt|)    (setq op |$hashOpApply|))
           ((eq op '|setelt|) (setq op |$hashOpSet|))
           (t                 (setq op (|hashString| (symbol-name op))))))
        (cond
         ((setq val
            (car
              (spadcall (cdr domain) dollar op hashSig box nil lookupFun)))
           val)
         ((|hashCode?| sig) nil)
         ((or (> (|#| sig) 1) (|opIsHasCat| op)) nil)
         ((setq boxval
          (spadcall (cdr dollar) dollar op
                    (|hashType| (car sig) hashPercent)
                    box nil lookupFun))
           (cons #'identity (car boxval)))
          (t nil)))
      ((|opIsHasCat| op) (|HasCategory| domain sig))
      (t
       (when (|hashCode?| op)
        (cond
         ((eql op |$hashOp1|)     (setq op '|One|))
         ((eql op |$hashOp0|)     (setq op '|Zero|))
         ((eql op |$hashOpApply|) (setq op '|elt|))
         ((eql op |$hashOpSet|)   (setq op '|setelt|))
         ((eql op |$hashSeg|)     (setq op 'segment))))
       (cond
        ((and (|hashCode?| sig) (eql sig hashPercent))
          (spadcall
           (car (spadcall (cdr dollar) dollar op '($) box nil lookupFun))))
        (t
         (car
```

```
            (spadcall (cdr dollar) dollar op sig box nil lookupFun))))))))))))
```

———-

69.12.4 defun lookupInDomainVector

[basicLookupCheckDefaults p1063]
[spadcall p??]

— **defun lookupInDomainVector** —

```
(defun |lookupInDomainVector| (op sig domain dollar)
 (if (consp domain)
   (|basicLookupCheckDefaults| op sig domain domain)
   (spadcall op sig dollar (elt domain 1))))
```

———-

69.12.5 defun basicLookupCheckDefaults

[vecp p??]
[error p??]
[hashType p??]
[hashCode? p??]
[hashString p??]
[spadcall p??]
[$lookupDefaults p??]

— **defun basicLookupCheckDefaults** —

```
(defun |basicLookupCheckDefaults| (op sig domain dollar)
 (declare (ignore domain))
 (let (box dispatch lookupFun hashPercent hashSig)
 (declare (special |$lookupDefaults|))
  (setq box (cons nil nil))
  (cond
   ((null (vecp (setq dispatch (car dollar))))
     (|error| '|bad domain format|))
   (t
     (setq lookupFun (elt dispatch 3))
     (cond
      ((eql (elt dispatch 0) 0)
        (setq hashPercent
         (if (vecp dollar)
```

```
          (|hashType| (elt dollar 0) 0)
          (|hashType| dollar 0)))
        (setq hashSig
         (if (|hashCode?| sig)
          sig
          (|hashType| (cons '|Mapping| sig) hashPercent)))
        (when (symbolp op) (setq op (|hashString| (symbol-name op))))
        (car (spadcall (cdr dollar) dollar op hashSig
                       box (null |$lookupDefaults|) lookupFun)))
      (t
        (car (spadcall (cdr dollar) dollar op sig box
                       (null |$lookupDefaults|) lookupFun))))))))
```

———-

69.12.6 defun oldCompLookup

[lookupInDomainVector p1063]
[$lookupDefaults p**??**]

— **defun oldCompLookup** —

```
(defun |oldCompLookup| (op sig domvec dollar)
 (let (|$lookupDefaults| u)
 (declare (special |$lookupDefaults|))
  (setq |$lookupDefaults| nil)
  (cond
   ((setq u (|lookupInDomainVector| op sig domvec dollar))
     u)
   (t
    (setq |$lookupDefaults| t)
    (|lookupInDomainVector| op sig domvec dollar)))))
```

———-

69.12.7 defun NRTevalDomain

[qcar p**??**]
[eval p**??**]
[evalDomain p903]

— **defun NRTevalDomain** —

```
(defun |NRTevalDomain| (form)
 (if (and (consp form) (eq (qcar form) 'setelt))
```

```
  (|eval| form)
  (|evalDomain| form)))
```

———-

69.13 Plot3d

We catch numeric errors and throw a different failure than normal. The trapNumericErrors macro will return a pair of the the form `Union(type-of-form, "failed")`. This pair is tested for eq-ness so it has to be unique. It lives in the defvar `$numericFailure`. The old value of the `$BreakMode` variable is saved in a defvar named `$oldBreakMode`.

69.13.1 defvar $numericFailure

This is a failed union branch which is the value returned for numeric failure.

— **initvars** —

```
(defvar |$numericFailure| (cons 1 "failed"))
```

———-

69.13.2 defvar $oldBreakMode

— **initvars** —

```
(defvar |$oldBreakMode| nil "the old value of the $BreakMode variable")
```

———-

69.13.3 defmacro trapNumericErrors

The following macro evaluates form returning Union(type-of form, "failed"). It is used in the `myTrap` local function in Plot3d.

— **defmacro trapNumericErrors** —

```
(defmacro |trapNumericErrors| (form)
 `(let ((|$oldBreakMode| |$BreakMode|) (|$BreakMode| '|trapNumerics|) (val))
  (declare (special |$BreakMode| |$numericFailure| |$oldBreakMode|))
   (setq val (catch '|trapNumerics| ,form))
   (if (eq val |$numericFailure|) val (cons 0 val))))
```

———-

69.14 DoubleFloatVector

Double Float Vectors are simple arrays of lisp double-floats made available at the Spad language level. Note that these vectors are 0 based whereas other Spad language vectors are 1-based.

69.14.1 defmacro dlen

DoubleFloatVector Qsize function support
— **defmacro dlen** —

```
(defmacro dlen (v)
 `(length (the (simple-array double-float (*)) ,v)))
```

———-

69.14.2 defmacro make-double-vector

DoubleFloatVector Qnew function support
— **defmacro make-double-vector** —

```
(defmacro make-double-vector (n)
 `(make-array (list ,n) :element-type 'double-float))
```

———-

69.14.3 defmacro make-double-vector1

DoubleFloatVector Qnew1 function support
— **defmacro make-double-vector1** —

```
(defmacro make-double-vector1 (n s)
 `(make-array (list ,n) :element-type 'double-float :initial-element ,s))
```

———-

69.14.4 defmacro delt

DoubleFloatVector Qelt1 function support
— **defmacro delt** —

```
(defmacro delt (v i)
 `(aref (the (simple-array double-float (*)) ,v) ,i))
```

———-

69.14.5 defmacro dsetelt

DoubleFloatVector Qsetelt1 function support
— **defmacro dsetelt** —

```
(defmacro dsetelt (v i s)
 `(setf (aref (the (simple-array double-float (*)) ,v) ,i) ,s))
```

———-

69.15 ComplexDoubleFloatVector

Complex Double Float Vectors are simple arrays of lisp double-floats made available at the Spad language level. Note that these vectors are 0 based whereas other Spad language vectors are 1-based. Complex array is implemented as an array of doubles. Each complex number occupies two positions in the real array.

69.15.1 defmacro make-cdouble-vector

ComplexDoubleFloatVector Qnew function support
— **defmacro make-cdouble-vector** —

```
(defmacro make-cdouble-vector (n)
   `(make-array (list (* 2 ,n)) :element-type 'double-float))
```

———-

69.15.2 defmacro cdelt

ComplexDoubleFloatVector Qelt1 function support
— **defmacro cdelt** —

```
(defmacro CDELT(ov oi)
   (let ((v (gensym))
         (i (gensym)))
   `(let ((,v ,ov)
          (,i ,oi))
      (cons
          (aref (the (simple-array double-float (*)) ,v) (* 2 ,i))
          (aref (the (simple-array double-float (*)) ,v) (+ (* 2 ,i) 1))))))
```

———-

69.15.3 defmacro cdsetelt

ComplexDoubleFloatVector Qsetelt1 function support
— **defmacro cdsetelt** —

```
(defmacro cdsetelt(ov oi os)
   (let ((v (gensym))
         (i (gensym))
         (s (gensym)))
   `(let ((,v ,ov)
          (,i ,oi)
          (,s ,os))
        (setf (aref (the (simple-array double-float (*)) ,v) (* 2 ,i))
           (car ,s))
        (setf (aref (the (simple-array double-float (*)) ,v) (+ (* 2 ,i) 1))
           (cdr ,s))
        ,s)))
```

———-

69.15.4 defmacro cdlen

ComplexDoubleFloatVector Qsize function support
— **defmacro cdlen** —

```
(defmacro cdlen(v)
    `(truncate (length (the (simple-array double-float (*)) ,v)) 2))
```

———-

69.16 DoubleFloatMatrix

69.16.1 defmacro make-double-matrix

DoubleFloatMatrix qnew function support
— **defmacro make-double-matrix** —

```
(defmacro make-double-matrix (n m)
   `(make-array (list ,n ,m) :element-type 'double-float))
```

———-

69.16.2 defmacro make-double-matrix1

DoubleFloatMatrix new function support
— **defmacro make-double-matrix1** —

```
(defmacro make-double-matrix1 (n m s)
   `(make-array (list ,n ,m) :element-type 'double-float
           :initial-element ,s))
```

———-

69.16.3 defmacro daref2

DoubleFloatMatrix qelt function support
— **defmacro daref2** —

```
(defmacro daref2 (v i j)
   `(aref (the (simple-array double-float (* *)) ,v) ,i ,j))
```

———-

69.16.4 defmacro dsetaref2

DoubleFloatMatrix qsetelt! function support
— **defmacro dsetaref2** —

```
(defmacro dsetaref2 (v i j s)
   `(setf (aref (the (simple-array double-float (* *)) ,v) ,i ,j)
          ,s))
```

———-

69.16.5 defmacro danrows

DoubleFloatMatrix nrows function support
— **defmacro danrows** —

```
(defmacro danrows (v)
    `(array-dimension (the (simple-array double-float (* *)) ,v) 0))
```

————-

69.16.6 defmacro dancols

DoubleFloatMatrix ncols function support
— **defmacro dancols** —

```
(defmacro dancols (v)
    `(array-dimension (the (simple-array double-float (* *)) ,v) 1))
```

————-

69.17 ComplexDoubleFloatMatrix

69.17.1 defmacro make-cdouble-matrix

ComplexDoubleFloatMatrix function support
— **defmacro make-cdouble-matrix** —

```
(defmacro make-cdouble-matrix (n m)
   `(make-array (list ,n (* 2 ,m)) :element-type 'double-float))
```

————-

69.17.2 defmacro cdaref2

ComplexDoubleFloatMatrix function support
— **defmacro cdaref2** —

```
(defmacro cdaref2 (ov oi oj)
   (let ((v (gensym))
         (i (gensym))
         (j (gensym)))
   `(let ((,v ,ov)
```

```
          (,i ,oi)
          (,j ,oj))
        (cons
            (aref (the (simple-array double-float (* *)) ,v) ,i (* 2 ,j))
            (aref (the (simple-array double-float (* *)) ,v)
                  ,i (+ (* 2 ,j) 1))))))
```

———-

69.17.3 defmacro cdsetaref2

ComplexDoubleFloatMatrix function support
— **defmacro cdsetaref2** —

```
(defmacro cdsetaref2 (ov oi oj os)
   (let ((v (gensym))
         (i (gensym))
         (j (gensym))
         (s (gensym)))
   `(let ((,v ,ov)
          (,i ,oi)
          (,j ,oj)
          (,s ,os))
         (setf (aref (the (simple-array double-float (* *)) ,v) ,i (* 2 ,j))
               (car ,s))
         (setf (aref (the (simple-array double-float (* *)) ,v)
                     ,i (+ (* 2 ,j) 1))
               (cdr ,s))
         ,s)))
```

———-

69.17.4 defmacro cdanrows

ComplexDoubleFloatMatrix function support
— **defmacro cdanrows** —

```
(defmacro cdanrows (v)
    `(array-dimension (the (simple-array double-float (* *)) ,v) 0))
```

———-

69.17.5 defmacro cdancols

ComplexDoubleFloatMatrix function support

— **defmacro cdancols** —

```
(defmacro cdancols (v)
    `(truncate
         (array-dimension (the (simple-array double-float (* *)) ,v) 1) 2))
```

———-

69.18 Integer

69.18.1 defun Integer divide function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses the Integer divide function actually inlines a call to this code. The Integer domain contains the line:

```
(PUT (QUOTE |INT;divide;2$R;44|) (QUOTE |SPADreplace|) (QUOTE DIVIDE2))
```

— **defun divide2 0** —

```
(defun divide2 (x y)
 (multiple-value-call #'cons (truncate x y)))
```

———-

69.18.2 defun Integer quo function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses the Integer quo function actually inlines a call to this code. The Integer domain contains the line:

```
(PUT (QUOTE |INT;rem;3$;46|) (QUOTE |SPADreplace|) (QUOTE REMAINDER2))
```

Because these are identical except for name we make the symbol-functions equivalent. This was done in the original code for efficiency.

— **defun remainder2 0** —

```
(setf (symbol-function 'remainder2) #'rem)
```

———-

69.18.3 defun Integer quo function support

Note that this is defined as a SPADReplace function in Integer so that algebra code that uses the Integer quo function actually inlines a call to this code. The Integer domain contains the line:

```
(PUT (QUOTE |INT;quo;3$;45|) (QUOTE |SPADreplace|) (QUOTE QUOTIENT2))
```

— **defun quotient2 0** —

```
(defun quotient2 (x y)
 (values (truncate x y)))
```

———-

69.18.4 defun Integer random function support

This is used for calls to random with no arguments. If an argument is supplied to random then the common lisp random function is called directly. This could be lifted up into the spad code.

— **defun random 0** —

```
(defun |random| () (random (expt 2 26)))
```

———-

69.19 IndexCard

69.19.1 defun IndexCard origin function support

[dbPart p**??**]
[charPosition p**??**]
[substring p**??**]

— **defun alqlGetOrigin** —

```
(defun |alqlGetOrigin| (x)
 (let (field k)
  (setq field (|dbPart| x 5 1))
  (setq k (|charPosition| #\( field 2))
  (substring field 1 (1- k))))
```

———-

69.19.2 defun IndexCard origin function support

[dbPart p??]
[charPosition p??]
[substring p??]

— **defun alqlGetParams** —

```
(defun |alqlGetParams| (x)
 (let (field k)
  (setq field (|dbPart| x 5 1))
  (setq k (|charPosition| #\( field 2))
  (substring field k nil)))
```

———-

69.19.3 defun IndexCard elt function support

[dbPart p??]
[substring p??]

— **defun alqlGetKindString** —

```
(defun |alqlGetKindString| (x)
 (if (or (char= (elt x 0) #\a) (char= (elt x 0) #\o))
  (substring (|dbPart| x 5 1) 0 1)
  (substring x 0 1))))
```

———-

69.20 OperationsQuery

69.20.1 defun OperationQuery getDatabase function support

This function, called as `getBrowseDatabase(arg)` returns a list of appropriate entries in the browser database. The legal values for arg are

- "o" (operations)
- "k" (constructors)
- "d" (domains)
- "c" (categories)

- "p" (packages)

[member p1042]
[grepConstruct p??]
[$includeUnexposed? p??]

— **defun getBrowseDatabase** —

```
(defun |getBrowseDatabase| (kind)
 (let (|$includeUnexposed?|)
 (declare (special |$includeUnexposed?|))
  (setq |$includeUnexposed?| t)
  (when (|member| kind '("o" "k" "c" "d" "p"))
   (|grepConstruct| "*" (intern kind)))))
```

———-

69.21 Database

69.21.1 defun Database elt function support

[basicMatch? p??]

— **defun stringMatches?** —

```
(defun |stringMatches?| (pattern subject)
 (when (integerp (|basicMatch?| pattern subject)) t))
```

———-

69.22 FileName

69.22.1 defun FileName filename function implementation

[StringToDir p1076]

— **defun fnameMake** —

```
(defun |fnameMake| (d n e)
  (if (string= e "") (setq e nil))
  (make-pathname :directory (|StringToDir| d) :name  n :type e))
```

———-

69.22.2 defun FileName filename support function

[lastc p??]

— **defun StringToDir** —

```
(defun |StringToDir| (s)
  (cond
    ((string= s "/") '(:root))
    ((string= s "")  nil)
    (t
      (let ((lastc (aref s (- (length s) 1))))
        (if (char= lastc #\/)
          (pathname-directory (concat s "name.type"))
          (pathname-directory (concat s "/name.type")) ))) ))
```

———-

69.22.3 defun FileName directory function implementation

[DirToString p1076]

— **defun fnameDirectory** —

```
(defun |fnameDirectory| (f)
  (|DirToString| (pathname-directory f)))
```

———-

69.22.4 defun FileName directory function support

For example, "/" "/u/smwatt" "../src"

— **defun DirToString 0** —

```
(defun |DirToString| (d)
  (cond
    ((equal d '(:root)) "/")
    ((null d) "")
    ('t (string-right-trim "/" (namestring (make-pathname :directory d)))) ))
```

———-

69.22.5 defun FileName name function implementation

— defun fnameName 0 —

```
(defun |fnameName| (f)
  (let ((s (pathname-name f)))
    (if s s "") ))
```

————-

69.22.6 defun FileName extension function implementation

— defun fnameType 0 —

```
(defun |fnameType| (f)
  (let ((s (pathname-type f)))
    (if s s "") ))
```

————-

69.22.7 defun FileName exists? function implementation

— defun fnameExists? 0 —

```
(defun |fnameExists?| (f)
  (if (probe-file (namestring f)) 't nil))
```

————-

69.22.8 defun FileName readable? function implementation

— defun fnameReadable? 0 —

```
(defun |fnameReadable?| (f)
 (let ((s (open f :direction :input :if-does-not-exist nil)))
  (cond (s (close s) t) (t nil)) ))
```

————-

69.22.9 defun FileName writeable? function implementation

[myWriteable? p**??**]

— **defun fnameWritable?** —

```
(defun |fnameWritable?| (f)
  (|myWritable?| (namestring f)) )
```

———-

69.22.10 defun FileName writeable? function support

[error p**??**]
[fnameExists? p1077]
[fnameDirectory p1076]
[writeablep p**??**]

— **defun myWritable?** —

```
(defun |myWritable?| (s)
  (if (not (stringp s)) (|error| "‘‘myWritable?’’ requires a string arg."))
  (if (string= s "") (setq s "."))
  (if (not (|fnameExists?| s)) (setq s (|fnameDirectory| s)))
  (if (string= s "") (setq s "."))
  (if (> (|writeablep| s) 0) 't nil) )
```

———-

69.22.11 defun FileName new function implementation

[fnameMake p1075]

— **defun fnameNew** —

```
(defun |fnameNew| (d n e)
  (if (not (|myWritable?| d))
    nil
    (do ((fn))
        (nil)
        (setq fn (|fnameMake| d (string (gensym n)) e))
        (if (not (probe-file (namestring fn)))
           (return-from |fnameNew| fn)) )))
```

———-

69.23 DoubleFloat

These macros wrap their arguments with strong type information in order to optimize doublefloat computatations. They are used directly in the DoubleFloat domain (see Volume 10.3).

69.23.1 defmacro DFLessThan

Compute a strongly typed doublefloat comparison See Steele Common Lisp 1990 p293

— **defmacro DFLessThan** —

```
(defmacro DFLessThan (x y)
 `(< (the double-float ,x) (the double-float ,y)))
```

———-

69.23.2 defmacro DFUnaryMinus

Compute a strongly typed unary doublefloat minus See Steele Common Lisp 1990 p295

— **defmacro DFUnaryMinus** —

```
(defmacro DFUnaryMinus (x)
 `(the double-float (- (the double-float ,x))))
```

———-

69.23.3 defmacro DFMinusp

Compute a strongly typed unary doublefloat test for negative See Steele Common Lisp 1990 p292

— **defmacro DFMinusp** —

```
(defmacro DFMinusp (x)
 `(minusp (the double-float ,x)))
```

———-

69.23.4 defmacro DFZerop

Compute a strongly typed unary doublefloat test for zero See Steele Common Lisp 1990 p292

— **defmacro DFZerop** —

```
(defmacro DFZerop (x)
 `(zerop (the double-float ,x)))
```

———-

69.23.5 defmacro DFAdd

Compute a strongly typed doublefloat addition See Steele Common Lisp 1990 p295

— **defmacro DFAdd** —

```
(defmacro DFAdd (x y)
 `(the double-float (+ (the double-float ,x) (the double-float ,y))))
```

———-

69.23.6 defmacro DFSubtract

Compute a strongly typed doublefloat subtraction See Steele Common Lisp 1990 p295

— **defmacro DFSubtract** —

```
(defmacro DFSubtract (x y)
 `(the double-float (- (the double-float ,x) (the double-float ,y))))
```

———-

69.23.7 defmacro DFMultiply

Compute a strongly typed doublefloat multiplication See Steele Common Lisp 1990 p296

— **defmacro DFMultiply** —

```
(defmacro DFMultiply (x y)
 `(the double-float (* (the double-float ,x) (the double-float ,y))))
```

———-

69.23.8 defmacro DFIntegerMultiply

Compute a strongly typed doublefloat multiplication by an integer. See Steele Common Lisp 1990 p296

— **defmacro DFIntegerMultiply** —

```
(defmacro DFIntegerMultiply (i y)
 `(the double-float (* (the integer ,i) (the double-float ,y))))
```

———-

69.23.9 defmacro DFMax

Choose the maximum of two doublefloats. See Steele Common Lisp 1990 p294

— **defmacro DFMax** —

```
(defmacro DFMax (x y)
 `(the double-float (max (the double-float ,x) (the double-float ,y))))
```

———-

69.23.10 defmacro DFMin

Choose the minimum of two doublefloats. See Steele Common Lisp 1990 p294

— **defmacro DFMin** —

```
(defmacro DFMin (x y)
 `(the double-float (min (the double-float ,x) (the double-float ,y))))
```

———-

69.23.11 defmacro DFEql

Compare two doublefloats for equality, where equality is eq, or numbers of the same type with the same value. See Steele Common Lisp 1990 p105

— **defmacro DFEql** —

```
(defmacro DFEql (x y)
 `(eql (the double-float ,x) (the double-float ,y)))
```

———-

69.23.12 defmacro DFDivide

Divide a doublefloat by a a doublefloat See Steele Common Lisp 1990 p296

— **defmacro DFDivide** —

```
(defmacro DFDivide (x y)
 `(the double-float (/ (the double-float ,x) (the double-float ,y))))
```

———-

69.23.13 defmacro DFIntegerDivide

Divide a doublefloat by an integer See Steele Common Lisp 1990 p296

— **defmacro DFIntegerDivide** —

```
(defmacro DFIntegerDivide (x i)
 `(the double-float (/ (the double-float ,x) (the integer ,i))))
```

———-

69.23.14 defmacro DFSqrt

Compute the doublefloat square root of x. The result will be complex if the argument is negative. See Steele Common Lisp 1990 p302

— **defmacro DFSqrt** —

```
(defmacro DFSqrt (x)
 `(sqrt (the double-float ,x)))
```

———-

69.23.15 defmacro DFLogE

Compute the doublefloat log of x with the base e. The result will be complex if the argument is negative. See Steele Common Lisp 1990 p301

— **defmacro DFLogE** —

```
(defmacro DFLogE (x)
 `(log (the double-float ,x)))
```

———-

69.23.16 defmacro DFLog

Compute the doublefloat log of x with a given base b. The result will be complex if x is negative. See Steele Common Lisp 1990 p301

— **defmacro DFLog** —

```
(defmacro DFLog (x b)
 `(log (the double-float ,x) (the fixnum ,b)))
```

———-

69.23.17 defmacro DFIntegerExpt

Compute the doublefloat expt of x with a given integer power i See Steele Common Lisp 1990 p300

— **defmacro DFIntegerExpt** —

```
(defmacro DFIntegerExpt (x i)
 `(the double-float (expt (the double-float ,x) (the integer ,i))))
```

———-

69.23.18 defmacro DFExpt

Compute the doublefloat expt of x with a given power p. The result could be complex if the base is negative and the power is not an integer. See Steele Common Lisp 1990 p300

— **defmacro DFExpt** —

```
(defmacro DFExpt (x p)
 `(expt (the double-float ,x) (the double-float ,p)))
```

———-

69.23.19 defmacro DFExp

Compute the doublefloat exp with power e See Steele Common Lisp 1990 p300

— **defmacro DFExp** —

```
(defmacro DFExp (x)
 `(the double-float (exp (the double-float ,x))))
```

———-

69.23.20 defmacro DFSin

Compute a strongly typed doublefloat sin See Steele Common Lisp 1990 p304

— **defmacro DFSin** —

```
(defmacro DFSin (x)
 `(the double-float (sin (the double-float ,x))))
```

———-

69.23.21 defmacro DFCos

Compute a strongly typed doublefloat cos See Steele Common Lisp 1990 p304

— **defmacro DFCos** —

```
(defmacro DFCos (x)
 `(the double-float (cos (the double-float ,x))))
```

———-

69.23.22 defmacro DFTan

Compute a strongly typed doublefloat tan See Steele Common Lisp 1990 p304

— **defmacro DFTan** —

```
(defmacro DFTan (x)
 `(the double-float (tan (the double-float ,x))))
```

———-

69.23.23 defmacro DFAsin

Compute a strongly typed doublefloat asin. The result is complex if the absolute value of the argument is greater than 1. See Steele Common Lisp 1990 p305

— **defmacro DFAsin** —

```
(defmacro DFAsin (x)
 `(asin (the double-float ,x)))
```

———-

69.23.24 defmacro DFAcos

Compute a strongly typed doublefloat acos. The result is complex if the absolute value of the argument is greater than 1. See Steele Common Lisp 1990 p305

— **defmacro DFAcos** —

```
(defmacro DFAcos (x)
 `(acos (the double-float ,x)))
```

———-

69.23.25 defmacro DFAtan

Compute a strongly typed doublefloat atan See Steele Common Lisp 1990 p305

— **defmacro DFAtan** —

```
(defmacro DFAtan (x)
 `(the double-float (atan (the double-float ,x))))
```

———-

69.23.26 defmacro DFAtan2

Compute a strongly typed doublefloat atan with 2 arguments

$y = 0$	$x > 0$	Positive x-axis	0
$y > 0$	$x > 0$	Quadrant I	$0 <$ result $< \pi/2$
$y > 0$	$x = 0$	Positive y-axis	$\pi/2$
$y > 0$	$x < 0$	Quadrant II	$\pi/2 <$ result $< \pi$
$y = 0$	$x < 0$	Negative x-axis	π
$y < 0$	$x < 0$	Quadrant III	$-\pi <$ result $< -\pi/2$
$y < 0$	$x = 0$	Negative y-axis	$-\pi/2$
$y < 0$	$x > 0$	Quadrant IV	$-\pi/2 <$ result < 0
$y = 0$	$x = 0$	Origin	error

See Steele Common Lisp 1990 p306

— **defmacro DFAtan2** —

```
(defmacro DFAtan2 (y x)
 `(the double-float (atan (the double-float ,x) (the double-float ,y))))
```

———-

69.23.27 defmacro DFSinh

Compute a strongly typed doublefloat sinh

$$(e^z - e^{-z})/2$$

See Steele Common Lisp 1990 p308

— **defmacro DFSinh** —

```
(defmacro DFSinh (x)
 `(the double-float (sinh (the double-float ,x))))
```

———-

69.23.28 defmacro DFCosh

Compute a strongly typed doublefloat cosh

$$(e^z + e^{-z})/2$$

See Steele Common Lisp 1990 p308

— **defmacro DFCosh** —

```
(defmacro DFCosh (x)
 `(the double-float (cosh (the double-float ,x))))
```

———-

69.23.29 defmacro DFTanh

Compute a strongly typed doublefloat tanh

$$(e^z - e^{-z})/(e^z + e^{-z})$$

See Steele Common Lisp 1990 p308

— **defmacro DFTanh** —

```
(defmacro DFTanh (x)
 `(the double-float (tanh (the double-float ,x))))
```

———-

69.23.30 defmacro DFAsinh

Compute the inverse hyperbolic sin.

$$log\left(z + \sqrt{1 + z^2}\right)$$

See Steele Common Lisp 1990 p308

— **defmacro DFAsinh** —

```
(defmacro DFAsinh (x)
 `(the double-float (asinh (the double-float ,x))))
```

———-

69.23.31 defmacro DFAcosh

Compute the inverse hyperbolic cos. Note that the acosh function will return a complex result if the argument is less than 1.

$$log\left(z + (z+1)\sqrt{(z-1)/(z+1)}\right)$$

See Steele Common Lisp 1990 p308

— **defmacro DFAcosh** —

```
(defmacro DFAcosh (x)
 `(acosh (the double-float ,x)))
```

———-

69.23.32 defmacro DFAtanh

Compute the inverse hyperbolic tan. Note that the acosh function will return a complex result if the argument is greater than 1.

$$log\left((1+z)\sqrt{1/(1-z^2)}\right)$$

See Steele Common Lisp 1990 p308

— **defmacro DFAtanh** —

```
(defmacro DFAtanh (x)
 `(atanh (the double-float ,x)))
```

———-

69.23.33 defun Machine specific float numerator

This is used in the DoubleFloat integerDecode function

— **defun integer-decode-float-numerator 0** —

```
(defun integer-decode-float-numerator (x)
 (integer-decode-float x))
```

———-

69.23.34 defun Machine specific float denominator

This is used in the DoubleFloat integerDecode function
— **defun integer-decode-float-denominator 0** —

```
(defun integer-decode-float-denominator (x)
 (multiple-value-bind (mantissa exponent sign) (integer-decode-float x)
  (declare (ignore mantissa sign)) (expt 2 (abs exponent))))
```

———-

69.23.35 defun Machine specific float sign

This is used in the DoubleFloat integerDecode function
— **defun integer-decode-float-sign 0** —

```
(defun integer-decode-float-sign (x)
 (multiple-value-bind (mantissa exponent sign) (integer-decode-float x)
  (declare (ignore mantissa exponent)) sign))
```

———-

69.23.36 defun Machine specific float bit length

This is used in the DoubleFloat integerDecode function
— **defun integer-decode-float-exponent 0** —

```
(defun integer-decode-float-exponent (x)
 (multiple-value-bind (mantissa exponent sign) (integer-decode-float x)
  (declare (ignore mantissa sign)) exponent))
```

———-

69.23.37 defun Decode floating-point values

This function is used by DoubleFloat to implement the "mantissa" and "exponent" functions.
— **defun manexp 0** —

```
(defun manexp (u)
  (multiple-value-bind (f e s)
    (decode-float u)
    (cons (* s f) e)))
```

———-

69.23.38 defun The cotangent routine

The cotangent function is defined as

$$cot(z) = \frac{1}{tan(z)}$$

— **defun cot 0** —

```
(defun cot (a)
  (if (or (> a 1000.0) (< a -1000.0))
    (/ (cos a) (sin a))
    (/ 1.0 (tan a))))
```

———-

69.23.39 defun The inverse cotangent function

The inverse cotangent (arc-cotangent) function is defined as

$$acot(z) = cot^{-1}(z) = tan^{-1}(\frac{1}{z})$$

See Steele Common Lisp 1990 pp305-307

— **defun acot 0** —

```
(defun acot (a)
  (if (> a 0.0)
    (if (> a 1.0)
       (atan (/ 1.0 a))
       (- (/ pi 2.0) (atan a)))
    (if (< a -1.0)
       (- pi (atan (/ -1.0 a)))
       (+ (/ pi 2.0) (atan (- a))))))
```

———-

69.23.40 defun The secant function

$$sec(x) = \frac{1}{cos(x)}$$

— **defun sec 0** —

```
(defun sec (x) (/ 1 (cos x)))
```

———-

69.23.41 defun The inverse secant function

$$asec(x) = acos\left(\frac{1}{x}\right)$$

— **defun asec 0** —

```
(defun asec (x) (acos (/ 1 x)))
```

———-

69.23.42 defun The cosecant function

$$csc(x) = \frac{1}{sin(x)}$$

— **defun csc 0** —

```
(defun csc (x) (/ 1 (sin x)))
```

———-

69.23.43 defun The inverse cosecant function

$$acsc(x) = \frac{1}{asin(x)}$$

— **defun acsc 0** —

```
(defun acsc (x) (asin (/ 1 x)))
```

———-

69.23.44 defun The hyperbolic cosecant function

$$csch(x) = \frac{1}{sinh(x)}$$

— **defun csch 0** —

```
(defun csch (x) (/ 1 (sinh x)))
```

———-

69.23.45 defun The hyperbolic cotangent function

$$coth(x) = cosh(x)csch(x)$$

— **defun coth 0** —

```
(defun coth (x) (* (cosh x) (csch x)))
```

———-

69.23.46 defun The hyperbolic secant function

$$sech(x) = \frac{1}{cosh(x)}$$

— **defun sech 0** —

```
(defun sech (x) (/ 1 (cosh x)))
```

———-

69.23.47 defun The inverse hyperbolic cosecant function

$$acsch(x) = asinh\left(\frac{1}{x}\right)$$

— **defun acsch 0** —

```
(defun acsch (x) (asinh (/ 1 x)))
```

———-

69.23.48 defun The inverse hyperbolic cotangent function

$$acoth(x) = atanh\left(\frac{1}{x}\right)$$

— **defun acoth 0** —

```
(defun acoth (x) (atanh (/ 1 x)))
```

———-

69.23.49 defun The inverse hyperbolic secant function

$$asech(x) = acosh\left(\frac{1}{x}\right)$$

— **defun asech 0** —

```
(defun asech (x) (acosh (/ 1 x)))
```

———-

Chapter 70

OpenMath

70.1 A Technical Overview[4]

OpenMath is a standard for representing mathematical data in as unambiguous a way as possible. It can be used to exchange mathematical objects between software packages or via email, or as a persistent data format in a database. It is tightly focussed on representing semantic information and is not intended to be used directly for presentation, although tools exist to facilitate this.

The original motivation for OpenMath came from the Computer Algebra community. Computer Algebra packages were getting bigger and more unwieldy, and it seemed reasonable to adopt a generic "plug and play" architecture to allow specialised programs to be used from general purpose environments. There were plenty of mechanisms for connecting software components together, but no common format for representing the underlying data objects. It quickly became clear that any standard had to be vendor-neutral and that objects encoded in OpenMath should not be too verbose. This has led to the design outlined below.

In 1998, the Worldwide Web Consortium (W3C) produced its first recommendation for the Extensible Markup Language (XML), intended to be a universal format for representing structured information on the worldwide web. It was swiftly followed by the first MathML recommendation which is an XML application oriented mainly towards the presentation (i.e. the rendering) of mathematical expressions.

The formal definition of OpenMath is contained within The OpenMath Standard and its accompanying documents, and the reader is referred there for more details.

70.1.1 The OpenMath Architecture

The OpenMath representation of a mathematical structure is referred to as an OpenMath object. This is an abstract structure which is represented concretely via an OpenMath encoding. These encoded objects are what an OpenMath application would read and write,

and in practice the OpenMath objects themselves almost never exist, except on paper. The advantage of this is that OpenMath is not tied to any one underlying mechanism: in the past we have used functional, SGML and binary encodings. The current favourite is XML, as described below, and we will tend to use XML notation when describing OpenMath objects (even though strictly speaking the XML representation is an encoding). OpenMath Objects

Formally, an OpenMath object is a labelled tree whose leaves are the basic OpenMath objects integers, IEEE double precision floats, unicode strings, byte arrays, variables or symbols. Of these, symbols are the most interesting since they consist of a name and a reference to a definition in an external document called a content dictionary (or CD). Using XML notation where the element name OMS indicates an OpenMath symbol, the following:

```
<OMS name="sin" cd="transc1"/>
```

represents the usual sine function, as defined in the CD "transc1". A basic OpenMath object is an OpenMath object, although its XML representation will be:

```
<OMOBJ>
  <OMS name="sin" cd="transc1"/>
</OMOBJ>
```

OpenMath objects can be built up recursively in a number of ways. The simplest is function application, for example the expression sin(x) can be represented by the XML:

```
<OMOBJ>
  <OMA>
    <OMS name="sin" cd="transc1"/>
    <OMV name="x"/>
  </OMA>
</OMOBJ>
```

where OMV introduces a variable and OMA is the application element. Another straightforward method is attribution which as the name suggests can be used to add additional information (for example "the AXIOM command which generated me was ...") to an object without altering its fundamental meaning. More interesting are binding objects which are used to represent an expression containing bound variables, for example:

```
<OMOBJ>
  <OMA>
    <OMS cd="calculus1" name="int"/>
    <OMS cd="transc1" name="sin"/>
  </OMA>
</OMOBJ>
```

represents the integral of the sin function, but the encoding:

```
<OMOBJ>
  <OMA>
```

```
    <OMS cd="calculus1" name="int"/>
    <OMBIND>
      <OMS cd="fns1" name="lambda"/>
      <OMBVAR> <OMV name="x"/> </OMBVAR>
      <OMA>
        <OMS name="sin" cd="transc1"/>
        <OMV name="x"/>
      </OMA>
    </OMBIND>
  </OMA>
</OMOBJ>
```

represents sin(x)dx. This may appear overly complicated but it is useful, for example when searching in a database for expressions which match sin(y)dy . The definition of a symbol in the CD specifies whether or not it may be used to bind variables, which is why

```
<OMS cd="calculus1" name="int"/>
```

cannot be used as a binding symbol.

The final kind of OpenMath object is an error which is built up from a symbol describing the error and a sequence of OpenMath objects. For example:

```
<OMOBJ>
  <OME>
    <OMS name="unexpected_symbol" cd="error1">
    <OMS name="sine" cd="transc1">
  </OME>
</OMOBJ>
```

represents the error which might be generated when an application sees a symbol it doesn't recognise from a CD it thought it knew about.

70.1.2 OpenMath Encodings

We have already seen some examples of the XML encoding, but it is by no means the only encoding. In the past there was a functional encoding (which looked like Lisp) and an SGML encoding which evolved into the current XML. Both of these are now obsolete, but there is still a binary encoding described in the standard , which is much more compact than the XML one.

In fact the XML encoding is not 100% XML. When XML was in its infancy the developers of OpenMath realised that it might become significant and decided to add some XML-like features to the SGML encoding so that an an OpenMath object could be encoded as valid XML. Thus it is currently the case that any well-formed OpenMath object encoded using the XML encoding as described in the standard is a valid XML document. However, if one uses standard XML tools to generate an OpenMath object in the XML encoding from the DTD given in chapter 4 of the standard, it is possible that the result will not be valid OpenMath,

although in practice this is highly unlikely. To cover all the possibilities allowed by XML would make it much more complicated to write an application to read any OpenMath object from scratch. Whether to adopt XML completely remains a hot topic of debate within the OpenMath community!

Generally speaking, it is not intended that the existing encodings should be readable by a human user or writable by hand. It is desirable that they be compact and it is also desirable that they be linear, but neither of these is a requirement. It is a property of encodings that it is possible to convert between them with no loss of information.

70.1.3 Content Dictionaries

Content Dictionaries (or CDs for short) are the most important, and the most interesting, aspect of OpenMath because they define the meaning of the objects being transmitted. A CD is a collection of related symbols and their definitions, encoded in an XML format. Defining the meaning of a symbol is not a trivial task, and even referring to well-known references can be fraught with pitfalls Formal definitions and properties can be very useful but time-consuming to produce and verbose, not to mention difficult to get right. A symbol definition in an OpenMath CD consists of the following pieces of information:

```
the symbol name;
a description in plain text;
optionally, a set of this symbol's properties in plain text
   (Commented Mathematical Properties, or CMPs);
optionally, a set of this symbol's properties encoded in OpenMath
   (Formal Mathematical Properties, or FMPs);
optionally, one or more examples of its use (encoded in OpenMath).
```

In practice the CMPs and FMPs can come as pairs, and often serve in the place of examples.

A very simple instance of a CD definition is:

```
<CDDefinition>
<Name> log </Name>

<Description>
This symbol represents a binary log function; the first argument is
the base, to which the second argument is log'ed.
It is defined in Abramowitz and Stegun, Handbook of Mathematical
Functions, section 4.1
</Description>
<CMP>
  a^b = c implies log_a c = b
</CMP>
<FMP>
  <OMOBJ>
    <OMA>
      <OMS cd="logic1" name="implies"/>
      <OMA>
```

```
        <OMS cd="relation1" name="eq"/>

        <OMA>
          <OMS cd="arith1" name="power"/>
          <OMV name="a"/>
          <OMV name="b"/>
        </OMA>
        <OMV name="c"/>
      </OMA>
      <OMA>
        <OMS cd="relation1" name="eq"/>

        <OMA>
          <OMS cd="transc1" name="log"/>
          <OMV name="a"/>
          <OMV name="c"/>
        </OMA>
        <OMV name="b"/>
      </OMA>
    </OMA>
  </OMOBJ>

</FMP>

<Example>
log 100 to base 10 (which is 2).
<OMOBJ>
  <OMA>
    <OMS cd="transc1" name="log"/>
    <OMF dec="10"/>
    <OMF dec="100"/>
  </OMA>
</OMOBJ>
</Example>

</CDDefinition>
```

Another example would be to print the list

```
   [ 1, 1/2 ]
```

as

```
<OMOBJ>
 <OMA>
  <OMS cd="list1" name="list"/>
  <OMI>1</OMI>
  <OMA>
   <OMS cd="nums1" name="rational"/>
   <OMI>1</OMI>
```

```
   <OMI>2</OMI>
  </OMA>
 </OMA>
</OMOBJ>
```

This provides a symbol to represent the log function by giving a pointer to a standard reference book. It provides the property that:

$$a^b = c \rightarrow log_a(c) = b$$

both as plain text and as OpenMath, and also gives an example of how the symbol is used.

CDs usually consist of related symbols and collections of related CDs can be grouped together, for convenience, as CD Groups. One very important CD Group is that corresponding to the content part of MathML.

It is possible to associate extra information with CDs, in particular type information. Since there are many type systems available, each of which has its own strengths and advocates, the OpenMath community does not mandate any single system. Simple signatures can be encoded using the Simple Type System, while more formal definitions are possible using the Extended Calculus of Constructorss. Other associated information can include style sheets for rendering OpenMath symbols in MathML, and mathematical definitions to be used by formal logic systems.

Given the evolutionary nature of mathematics, it is clear that the set of CDs should be forever growing and never complete. Currently there are CDs for high-school mathematics, linear algebra, polynomials and group theory to name a few, and new contributions are always welcome. There is no requirement that applications use the standard set of CDs and it is often very useful to design a "private" CD for a specific purpose.

70.1.4 OpenMath in Action

There is no definitive way in which OpenMath should be used, as the protocol has been designed to be as flexible as possible. Nevertheless many OpenMath applications share common characteristics which we shall discuss here.

Suppose that we wish to have two applications communicating by sending OpenMath objects to each other, e.g. a client program and a computational server. It is unlikely that the internal data structures used by the applications will be OpenMath, and so translation between the internal representations and OpenMath (almost certainly OpenMath encodings rather than objects) will have to take place. The piece of software which does this is usually referred to as a phrase-book.

It is possible to write a generic phrase-book which can handle any piece of OpenMath, but applications where this makes sense are few and far between. In practice an OpenMath phrase book will usually only handle a fixed set of CDs (and hence a fixed set of symbols). What "handle" means will vary from case to case: a computer algebra system will usually try and evaluate its input and return a result or an error, while a typesetter will print its input according to some rendering rules and not return anything. OpenMath carefully avoids

defining what the "right" behaviour is in a given circumstance, and leaves that up to the phrase-book writer. Indeed it is quite possible that a piece of software could have multiple phrase-books associated with it for different purposes. OpenMath symbols should not be regarded as verbs since they are used to construct objects rather than to send commands, and the presence of both nouns and verbs in a CD (e.g. "integral" and "integrate") is strongly discouraged.

Writing a phrase-book may be non-trivial, and requires an understanding of the semantics of the underlying software. An OpenMath object may not map directly into a private object and vice-versa, for example in some systems a rational number might have to be represented by a float, or a sparse matrix by a dense one.

The OpenMath standard includes a section on compliance, which describes the behaviour of an OpenMath application when certain errors occur. It also insists that all compliant software has the capability to use the XML encoding, to guarantee a degree of interoperability. This is an area where the standard is expected to evolve as more OpenMath applications become available.

70.2 Technical Details[3]

This chapter describes the Axiom implementation of the OpenMath project at INRIA [3]. The code enables the exchange of OpenMath objects between two processes and more generally the input and output of OpenMath objects. First we describe the library API and then we implement the functions used by Axiom.

70.3 The Structure of the API

The library and its API are logically structured in four parts:

- Functions that deal with *devices*, the abstraction from which OpenMath objects are read and written to.
- Functions that read from and write to OpenMath devices. These functions use a simple model that read and write tokens.
- Functions that create I/O structures to be used by devices, so that, for example, an OpenMath object can be read from a file or a socket. This part is extensible by the user.
- Functions that deal with interprocess communication.

70.4 OpenMath Expressions

70.4.1 Expressions

The library understands the following kinds of basic OpenMath expressions:

- integers
- double precision floating-point numbers (64 bits, following IEEE 754)
- byte arrays
- character strings
- symbols
- variables

and the four kinds of constructions:

- applications $e_0(e_1, \ldots e_n)$
- errors $s(e_1, \ldots e_n)$
- binders $e_1, (v_1, \ldots v_n), e_2$
- attributed expressions $[s_1 e_1, \ldots s_n e_n]e$

where e_i are OpenMath expressions, v_i are OpenMath variables and s and s_i are OpenMath symbols.

70.4.2 Symbols

Symbols are constructed from a content dictionary (abbreviated as CD in the sequel) and a name. A content dictionary is identified by its name. The API permits the creation of any symbol in any content dictionary: there is nothing that prevents creating symbols that do not belong to a known CD.

70.4.3 Encoding and Decoding OpenMath Expressions

An OpenMath object is encoded as a sequence of bytes that is read and written sequentially. The library views this sequence as a stream of tokens. Expressions are linearized in a way that looks like Lisp with typed parenthesis. For example, the linearization of the application of S to $E_1 \ldots E_n$ is:

- indicating that this is an application (a "begin application" token)
- linearizing S

- linearizing E_1,... E_n
- indicating that all arguments have been given (an "end application" token)

The other constructions are linearized the same way (each one with its own begin and end tokens). Note that there is no explicit arity indication so that we don't have to introduce a special mechanism when we don't know beforehand how many arguments there are.

To give attributes to an expression, the attributes and their associated values are put before the expression. To give the attributes a_i with values v_i (where a_i are symbols and v_i are OpenMath expressions) to an expression E the process is:

- put a "begin attributed expression" token
- put a "begin attribute pairs" token
- put the symbol a_1 followed by the linearization of v_1 *etc*
- put an "end attribute pairs" token
- linearize E
- put an "end attributed expression" token

Decoding is done by first querying the type of the next OpenMath token and then invoking the right function to get this particular kind of token.

70.5 Big Integers

The library supports big integers that can potentially be given in various formats. The `OMBigIntType` describes the different possible formats.

```
typedef enum OMbigIntType {
  OMBIunknown = 0,  /* this is base 10, digits in normal order */
  OMBIbase10        /* this is base 16, digits in normal order (MSB) */
  OMbigIntBase16
} OMbigIntType;
```

70.6 Functions Dealing with OpenMath Devices

OpenMath expressions are read and written through *devices*. Basically, an OpenMath device has an associated encoding and an I/O method. There are basically two encodings defined and implemented. The first one is a human readable and writable one that can be used for example as the encoding for sending OpenMath objects via e-mail or storing OpenMath objects to files. This encoding is SGML compatible in the sense that it can be used to represent OpenMath objects in SGML texts. I has an XML variant. The second encoding

is a binary one that can be used when compactness and speed of encoding and decoding is important. The encodings are defined by the `OMencodingType` type which is an enumerated type defined as

```
typedef enum OMencodingType
   { OMencodingUnknown,
     OMencodingBinary,
     OMencodingSGML,
     OMencodingXML} OMencodingType;
```

`OMencodingUnknown` is to be used when creating a device that does not know which kind of encoding will be used. It must be used only for input devices.

A device is created with the following function, given an encoding and an appropriate I/O method:

- `OMdev OMmakeDevice(OMencodingType encoding, OMIO IO)`

Devices are closed with the following function

- `void OMcloseDevice(OMdev dev)`

Whether a device could be used both for reading and writing is entirely dependent on its I/O method.

The user can define its own I/O method as a function returning an `OMIO` object. This could enable him, for example, to use an existing transport protocol to exchange OpenMath expressions or to implement cut-and-paste of OpenMath expression by writing I/O structures that input and output to strings. The I/O section describes the available I/O structures in the library.

An `OMdev` object is a pointer to a structure that contains a lot of state. Almost all functions taking an `OMdev` object modify it. Likewise, an `OMIO` object carries a lot of state.

70.7 Functions to Write OpenMath Expressions to Devices

70.7.1 Beginning and Ending Objects

The following two functions mark the beginning and end of an OpenMath object.

- `OMstatus OMputObject(OMdev dev)`
- `OMstatus OMputEndObject(OMdev dev)`

These functions should be called before and after an OpenMath object in constructed in a device. In particular, the `OMputEndObject` function insures that the object has been completely written if any buffering was used.

70.7.2 Writing Basic Objects

Basic OpenMath objects are written using these functions:

- `OMstatus OMputInt32(OMdev dev, int n)`
- `OMstatus OMputBigInt(OMdev dev, const char *data, int len, int sign, OMbigIntType format)`
- `OMstatus OMputFloat64(OMdev dev, double *f)`
- `OMstatus OMputByteArray(OMdev dev, const char *data, int len)`
- `OMstatus OMputString(OMdev dev, const char *s)`
- `OMstatus OMputVar(OMdev dev, const char *name)`
- `OMstatus OMputSymbol(OMdev dev, const char *cd, const char *name)`

The `char *` arguments of `OMputString`, `OMputVar` and `OMputSymbol` are null-terminated strings. There are other functions that accept non null-terminated arrays of characters with their length. These are

- `OMstatus OMputStringN(OMdev dev, const char *str, int len)`
- `OMstatus OMputVarN(OMdev dev, const char *var, int len)`
- `OMstatus OMputSymbolN(OMdev dev, const char *cd, int clen, const char *name, int nlen)`

The format for the `data` argument of the `OMputBigInt` function is given by `format`. When `format` is `OMBIbase10`, it is the sequence of character of its base 10 representation without sign (most significant digit first). The sign of the big integer is given by the `sign` argument that should be an integer greater or equal to zero for a positive integer and less than zero for a negative one. For example, the following line outputs the value of 20! to `dev`:

```
OMputBigInt(dev, "2652528598121910586363084800000000", 33, 1, OMBIbase10);
```

70.7.3 Writing Structured Objects

The following functions are used to mark the beginning and end of the structured objects. They should be called in nested pairs, correctly bracketed:

- `OMstatus OMputApp(OMdev dev)`
- `OMstatus OMputEndApp(OMdev dev)`
- `OMstatus OMputAttr(OMdev dev)`
- `OMstatus OMputEndAttr(OMdev dev)`

- `OMstatus OMputBind(OMdev dev)`
- `OMstatus OMputEndBind(OMdev dev)`
- `OMstatus OMputBVar(OMdev dev)`
- `OMstatus OMputEndBVar(OMdev dev)`
- `OMstatus OMputAtp(OMdev dev)`
- `OMstatus OMputEndAtp(OMdev dev)`
- `OMstatus OMputError(OMdev dev)`
- `OMstatus OMputEndError(OMdev dev)`

Here is an example showing how to use these functions to output $\sin x + y$, where x and y are represented as variables and sin is the symbol whose name is `sin` in the `Basic` content dictionary. This can be done using the following sequence:

```
OMputObject(dev);
OMputApp(dev);
  OMputSymbol(dev, "Basic", "sin");
  OMputApp(dev)
    OMputSymbol(dev, "Basic", "+");
    OMputVar(dev, "x");
    OMputVar(dev, "y");
  OMputEndApp(dev);
OMputEndApp(dev);
OMputEndObject(dev);
```

70.8 Functions to Extract OpenMath Expressions from Devices

70.8.1 Testing the type of the current token

The first step in decoding an expression from a device is to call the `OMgetType` function

- `OMstatus OMgetType(OMdev dev, OMtokenType *type)`

so that the correct function can be called to recover the current token.

`OMgetType` returns via its `type` argument an `OMtokenType` object indicating the type of the next object to be read from the device. `OMtokenType` is an enumerated type defined as

```
typedef enum OMtokenType {
  OMtokenUnknown, /* error catching trick */
  OMtokenInt32,
```

```
  OMtokenBigInt,
  OMtokenFloat64,
  OMtokenByteArray,
  OMtokenVar,
  OMtokenString,
  OMtokenSymbol,
  OMtokenComment,
  OMtokenApp,    OMtokenEndApp,
  OMtokenAttr,   OMtokenEndAttr,
  OMtokenAtp,    OMtokenEndAtp,
  OMtokenError,  OMtokenEndError,
  OMtokenObject, OMtokenEndObject,
  OMtokenBind,   OMtokenEndBind,
  OMtokenBVar,   OMtokenEndBVar,
} OMtokenType;
```

Note that the type of the current token can be tested multiple times. Two successive calls to `OMgetType` will always return the same result if no other `OMget...` function was called in between.

70.8.2 Extracting the current token

The following functions are used to read the basic OpenMath objects from devices:

- `OMstatus OMgetInt32(OMdev dev, int *i)`
- `OMstatus OMgetFloat64(OMdev dev, double *d)`
- `OMstatus OMgetBigInt(OMdev dev, char **data, int *len, int *sign, OMbigIntType *fmt)`
- `OMstatus OMgetBigIntN(OMdev dev, char *data, int len, int *sign, OMbigIntType *fmt)`
- `OMstatus OMgetByteArray(OMdev dev, char **data, int *len)`
- `OMstatus OMgetByteArrayN(OMdev dev, char *data, int len)`
- `OMstatus OMgetString(OMdev dev, char **str)`
- `OMstatus OMgetStringN(OMdev dev, char *str, int len)`
- `OMstatus OMgetVar(OMdev dev, char **var)`
- `OMstatus OMgetVarN(OMdev dev, char *var, int len)`
- `OMstatus OMgetSymbol(OMdev dev, char **cd, char **name)`
- `OMstatus OMgetSymbolN(OMdev dev, char *cd, int clen, char *name, int nlen)`

The functions that return variable size data exist in two versions. A simple version that does the necessary memory allocation itself (using `OMmalloc`) and a version (suffixed with `N`) that lets the user do the allocation itself. The size of the needed area can be determined with the following function:

- `int OMgetLength(OMdev dev)` returns the length of the next object.

that works for big integers, byte arrays, strings and variables. For symbols, the following function returns both the length of the content dictionary name and the length of the symbol name:

- `OMstatus OMgetSymbolLength(OMdev dev, int *clen, int *nlen)`

When the current token does not carry any data i.e. when `OMgetType` returns a marker, i.e. one of:

- `OMtokenApp`,
- `OMtokenEndApp`,
- `OMtokenAttr`,
- `OM tokenEndAttr`,
- `OMtokenAtp`,
- `OMtokenEndAtp`,
- `OMtokenError`,
- `OMtokenEndError`,
- `OMtokenObject`,
- `OMtokenEndObject`,
- `OMtokenBind`,
- `OMtokenEndBind`,
- `OMtokenBVar`
- `OMtokenEndBVar`

it is necessary to call the correct function to remove the marker. The available functions are

- `OMstatus OMgetObject(OMdev dev)`
- `OMstatus OMgetEndObject(OMdev dev)`
- `OMstatus OMgetApp(OMdev dev)`

- `OMstatus OMgetEndApp(OMdev dev)`
- `OMstatus OMgetAttr(OMdev dev)`
- `OMstatus OMgetEndAttr(OMdev dev)`
- `OMstatus OMgetAtp(OMdev dev)`
- `OMstatus OMgetEndAtp(OMdev dev)`
- `OMstatus OMgetBind(OMdev dev)`
- `OMstatus OMgetEndBind(OMdev dev)`
- `OMstatus OMgetBVar(OMdev dev)`
- `OMstatus OMgetEndBVar(OMdev dev)`
- `OMstatus OMgetError(OMdev dev)`
- `OMstatus OMgetEndError(OMdev dev)`

All the previous functions return `OMsuccess` when they succeed. When they return something else, there has been a problem such as calling the wrong function (`OMgetApp` when there is not a "beginning of application" mark) or a system error.

The sequence of calls to read an expression is thus completely similar (if we omit the calls to `OMgetType`) to the sequence of calls to write the expression. For example, the previous expression ($\sin x + y$) can be recovered via the sequence:

```
OMgetObject(dev);
OMgetApp(dev);
  OMgetSymbol(dev, ...);
  OMgetApp(dev);
    OMgetSymbol(dev, ...);
    OMgetVar(dev, ...);
    OMgetVar(dev, ...);
  OMgetEndApp(dev);
OMgetEndApp(dev);
OMgetEndObject(dev);
```

`OMgetInt32(OMdev dev, int *i)` returns the integer through its `i` argument.

```
OMgetBigInt(OMdev dev, char **data, int *len, int *sign, OMbigIntType *fmt)
```

returns the data corresponding to the big integer in `data`, its length in `len`, its sign in `sign` and its format in `fmt`.

```
OMgetBigIntN(OMdev dev, char *data, int len, int *sign, OMbigIntType *fmt)
```

copies the data corresponding to the big integer in `data` buffer that should be (at least) `len` characters long. The sign and format are returned in the `sign` and `fmt` arguments.

`OMgetByteArray(OMdev dev, char **data, int *len)` returns the byte array through its `data` argument. Its length is returned via the `len` argument.

`OMgetByteArrayN(OMdev dev, char *data, int len)` copies the byte array in the `data` buffer that should be (at least) `len` characters long.

`OMgetString(OMdev dev, char **str)` returns the string trough its `str` argument.

`OMgetStringN(OMdev dev, char *str, int len)` copies the string in the `str` buffer whose length should be (at least) `len`. If `len` is greater than the actual length of the string, a null character is added at the end of `str`.

`OMgetVar(OMdev dev, char **var)` returns the name of the variable (as a null-terminated string) in its `var` argument

`OMgetVarN(OMdev dev, char *var, int len)` copies the name of the variable in the `var` buffer, whose length should be (at least) `len`. If `len` is greater than the actual length of the variable name, a null character is added at the end of `var`.

`OMgetSymbol(OMdev dev, char **cd, char **name)` returns the content dictionary and the name of the symbol through the `cd` and `name` arguments.

`OMgetSymbolN(OMdev dev, char *cd, int clen, char *name, int nlen)` copies the content dictionary and the name of the symbols in the `cd` and `name` buffers. `cd` should be at least `clen` character long and `name` should be at least `nlen` long. When there is enough room (based on `clen` or `nlen`) a null character is added after the last character of the name (`cd` or `name`).

70.9 Comments in the SGML/XML Encodings

The library can also output and read comments (SGML/XML comments) with the following functions:

- `OMstatus OMputComment(OMdev dev, char *comment)`
- `OMstatus OMputCommentN(OMdev dev, char *comment, int len)`
- `OMstatus OMgetComment(OMdev dev, char **comment)`
- `OMstatus OMgetCommentN(OMdev dev, char *comment, int len)`

By default, comments are silently ignored by the library when reading OpenMath objects (and writing them using the binary encoding). The function

- `OMbool OMignoreComment(OMdev dev, OMbool set)`

changes this behaviour. When called with `OMfalse`, comments are passed to the application: the `OMgetType` function will return `OMtokenComment` when the current token is a comment and the `OMgetComment` or `OMgetCommentN` functions should be used to get the comments. When `OMignoreComment` is called with `OMtrue`, comments are ignored.

70.10 I/O Functions for Devices

We provide four functions that produce `OMIO` objects for devices. These functions provide I/O through the `stdio` library (on `FILE` object), file descriptors and character strings.

- `OMIO OMmakeIOFile(FILE *f)` associates the device with the file pointer `f`.
- `OMIO OMmakeIOfd(int fd)` associates the device with the file descriptor `fd`.
- `OMIO OMmakeIOHandle(HANDLE handle)` associates the device with a file handle *Windows specific version of `OMmakeIOfd().fd`.
- `OMIO OMmakeIOString(char **s)` associates the device with a string.

For example, the following code opens a device that reads from standard input:

```
dev = OMmakeDevice(OMencodingSGML, OMmakeIOFile(stdin));
```

The `OMmakeIOString` builds an input device that reads from a string or an output device that writes to a string. For input, `s` must point to a character string (null terminated). For output, `s` will point to a string allocated by the library (note that the string `s` points to can be reallocated by the library).

70.11 Communications

A communication layer can be put above the device layer. In fact, the I/O structure in a device provides all the necessary support to use any transmission or communication means. This library directly provides some connection-oriented, client-server facilities (based on TCP).

A set of functions are used to set up connections. Connections are described by the `OMconn` type. An `OMconn` is a (pointer to a) structure with two user-accessible fields `in` and `out`. `in` is a pointer to a device to be used for input. `out` is pointer to a device to be used for output. These devices use the binary encoding.

An `OMconn` object is made with the following function:

- `OMconn OMmakeConn(int timeout)`

where `timeout` is a timeout for the connection, expressed in milliseconds.

- `OMdev OMconnIn(OMconn conn)` returns the input device associated with the connection.
- `OMdev OMconnOut(OMconn conn)` returns the output device associated with the connection.

70.11.1 Functions to Initiate an `OMconn`

The functions we provide can be divided in two classes. The first one simply establishes an interprocess communication using IP addresses. The second one provides functions that can be used to launch a server. The addresses used are then generated by the library.

Simple Connections Functions

The following functions allow a client OpenMath application to contact an OpenMath server at a specified address:

- `OMstatus OMconnTCP(OMconn conn, char *machine, int port)`
- `OMstatus OMconnUnix(OMconn conn, char *file)`

These functions first physically establish the connection. Then, they enter negotiation with the server (they send the first message). When they return, the negotiation is finished and the devices in the `conn` argument are ready.

On the server side, the following functions provide bindings at specified addresses and take care of the negotiation:

- `OMstatus OMbindTCP(OMconn conn, int port)`
- `OMstatus OMbindUnix(OMconn conn, char *file)`

All four the previous functions block until the connection is established (and negotiation is over) or the timeout of the `conn` argument is reached.

The following function returns the file descriptor associated with a device. This is intended to be used when there is a need to poll the device (through the `select` or `poll` system calls).

- `OMdeviceFd(OMdev dev)`

Functions that Launch Servers

These functions provide the same functionalities for launching a server that were provided in the ASAP library.

In this model, the client calls `OMlaunch` with a machine name `mach` and a string `cmd` that is executed via `rsh` on machine `mach` as a shell command line. This command is supposed to launch the server program. The command is executed in an environment (in the UNIX sense) where some variables are associated with an address on the machine that runs the client. The server can then connect to the client with the `OMserveClient` function.

If the machine name is `localhost`, the command is started on the same machine (without calling `rsh`).

- `OMstatus OMlaunchEnv(OMconn conn, char *machine, char *command, char *env)`

- `OMstatus OMlaunch(OMconn conn, char *machine, char *command)`
- `OMstatus OMserveClient(OMconn conn)`

The environment variables passed to the server (launched program) are `OM_CALLER_UNIX_SOCKET` (when a local connection is required) and `OM_CALLER_MACHINE` and `OM_CALLER_PORT` (for internet connections).

The `OMlaunchEnv` function enables the command to be run with a particular environment (in the UNIX sense). For example to run a `plot` server on the `kama` machine, we could use a piece of code such as

```
conn = OMmakeConn(2000);
OMlaunchEnv(conn, "kama", "plot", "DISPLAY=rati:0 PATH=/users/bin");
```

Termination

- `OMstatus OMconnClose(OMconn conn)`

70.12 Parameters

The library internally uses three functions that can be supplied by the user.

- `extern void *(*OMmalloc) (size_t size)`
- `extern void *(*OMrealloc) (void *ptr, size_t size)`
- `extern void (*OMfree) (void *ptr)`

- `OMmalloc` is used for all memory allocations in the library. The default value is the `malloc` function.
- `OMfree` is used for deallocations. The default value is the `free` function.
- `OMfatal` is invoked when a fatal error is detected in the library (for example when memory allocation failed or when an inconsistency is detected in the library code data structures). The default value just does an `exit`.

`OMfatal` is declared as `extern void (*OMfatal)(OMstatus status)`. All memory allocations and deallocations in the library are done through the `OMmalloc` and `OMfree` functions.

70.13 Miscellaneous Functions and Variables

- `char *OMstatusToString(OMstatus status)` converts a status to a human readable string.

- `char *OMtokenTypeToString(OMtokenType ttype)` converts a `tokenType` to a human readable string.
- `OMencodingType OMgetDeviceEncoding(OMdev dev)` returns the encoding actually used by the device.
- `char *OMlibDynamicInfo(void)`
- `extern const char *OMlibVersion` is the version of the library.
- `extern const char *OMlibInfo` contains some textual information about the library.

70.14 The OM.h header file

```
#ifndef __OM_h__
#define __OM_h__

/*
 *                      All types used through API.
 */

/* These types are anonymized by the mean of a generic pointer.
 * You should not allocate or dereference objects of these types.
 * API (hopefully) provides you with all needed methods.
 * If you find any that are not included, please refer to
 * us rather than using private structures.
 * ie: If you need to do something like
 *   malloc(sizeof(OMdevStruct));
 * or
 *   OMdevStruct * pDev;
 *   pDev->anyField = something;
 * this probably means we need to discuss your problem.
 */

/* A device is an abstraction for put/get of OpenMath tokens */
typedef struct OMdevStruct *OMdev;

/* IO is a device field, (the physical IO channel) */
typedef struct OMIOStruct *OMIO;


/* Error status that may be returned
 */
typedef enum OMstatus {
  /* Last call was successful. */
  OMsuccess = 0,
  /* Last call failed for some undetermined reason. */
  OMfailed = 1,
  /* Last call failed for memory reasons. */
```

```
  OMnoMem,
  /* Last call failed during some system call. */
  OMerrorSys,
  /* Last call to some OMget* function failed due to an unexpected EOF
     on input IO. */
  OMemptyIO,
  /* Last call to some OMget* function failed because there is no more
     token on device. */
  OMnoMoreToken,
  /* Last call to some OMget* function timeouted. */
  OMtimeoutedRead,
  /* Last call to some OMget* function failed due to malformed input.
    (this error covers all low level lexical or syntactic problems). */
  OMmalformedInput,
  /* Last call to OMbindTCP failed because address is already in use
     (EADDRINUSE). */
  OMaddrInUse,
  /* Last call to OMconnTCP failed to set connection. */
  OMconnectFailed,
  /* Last call trigered some not (yet) implemented code in this lib. */
  OMnotImplemented,
  /* Last call caused some internal trouble. */
  OMinternalError
} OMstatus;

/* All OpenMath token kinds are identified by one of these types.
 * Values given in this enum have been chosen to:
 * - avoid  conflicts whith specific XML characters
 *   to help automatic detection of encoding type.
 *   (no: '\t'(9) '\r'(13) '\n'(10) '<'(60) or ' '(32))
 * - keep some bits (3) available for special encodings purpose
 *   (eg: sharing or big len flags in binary encoding)
 */
typedef enum OMtokenType {
  OMtokenUnknown = 0, /* error catching trick */
  OMtokenInt32 = 1,
  OMtokenBigInt = 2,
  OMtokenFloat64 = 3,
  OMtokenByteArray = 4,
  OMtokenVar = 5,
  OMtokenString = 6,
  OMtokenWCString = 7,
  OMtokenSymbol = 8,
  OMtokenComment = 15,
  OMtokenApp = 16, OMtokenEndApp = 17,
  OMtokenAttr = 18, OMtokenEndAttr = 19,
  OMtokenAtp = 20, OMtokenEndAtp = 21,
  OMtokenError = 22, OMtokenEndError = 23,
  OMtokenObject = 24, OMtokenEndObject = 25,
  OMtokenBind = 26,   OMtokenEndBind = 27,
```

```
  OMtokenBVar = 28,   OMtokenEndBVar = 29
} OMtokenType;

typedef enum OMbigIntType {
  OMbigIntUnknown = 0,
  /* this is base 10, digits in normal order (MSB) */
  OMbigIntBase10,
  /* this is base 16, digits in normal order (MSB) */
  OMbigIntBase16
} OMbigIntType;

/* Encodings should not be "user visible"
 * We thus refer to encoding as "symbolic constants" from this enum type. */
typedef enum OMencodingType {
  /* You may set an input stream to "unknown encoding".
   * By doing this, you let library auto detect the
   * encoding type of the device during first token input.*/
  OMencodingUnknown = 0,
  /* Binary encoding, more compact than XML one. */
  OMencodingBinary,
  /* XML-like encoding, human readable. */
  OMencodingXML,
} OMencodingType;

/* This is a portable equivalent to wchar_t for unicode strings */
typedef unsigned short OMUCS2;

/* Replacment for lacking C bools */
typedef unsigned char OMbool;
#define OMfalse (0)
#define OMtrue  (1)

/*
 *                        Some global variables
 */

/* Version of this lib (eg: "1.0") */
extern const char *OMlibVersion;

/* Some textual information about this lib (eg: "debug is on" */
extern const char *OMlibInfo;

/* These pointers allow you to redefine memory managment functions
   used in lib. */
extern void *(*OMmalloc) (size_t size);
extern void *(*OMrealloc) (void *ptr, size_t size);
extern void (*OMfree) (void *ptr);

/* If set, this function will be called by OMfatal, thus you may use it for
   error handling (by default it is set to exit()) */
```

```
extern void (*OMfatal) (OMstatus status);



/* for C++ includes */
#ifdef  __cplusplus
#define OMbeginPrototypes   extern "C" {
#define OMendPrototypes     }
#else /*__cplusplus */
#define OMbeginPrototypes
#define OMendPrototypes
#endif /*__cplusplus */

/*
 *                       Prototypes of OpenMath API
 */

/* Prototypes that are spread along all headers are repeated here.
 * - This should ease the API users.
 *   (docs are fine but source is always the ultimate help)
 * - This allow a cleaner embeding of library
 *   (no need to install all .h! just take this one and the .a)
 */
OMbeginPrototypes
#ifndef OM_DEV
/* this part is automaticaly updated, do NOT edit below */
/** Prototypes */
/* OMPut* functions.
 *    They all take a device <dev> to put token to.
 *    Some of them need more parameters to define the token content.
 *    They are thoroughly documented in OpenMath Specification shiped
 *    with the library.
 * return: a status that reflect the operation success.
 */
extern OMstatus OMputInt32(OMdev dev, int n);
extern OMstatus OMputFloat64(OMdev dev, double *d);
extern OMstatus OMputBigInt(OMdev dev, const char *data, int len,
                            int sign, OMbigIntType format);
extern OMstatus OMputByteArray(OMdev dev, const char *data, int len);
/* OMputString*
 *   If you want to output plain 8bits C like strings there is no need
 *   to use the OMputWCString* functions. This one is more efficient
 *   (faster and more compact output for some encodings)
 */
extern OMstatus OMputString(OMdev dev, const char *str);
extern OMstatus OMputStringN(OMdev dev, const char *str, int len);
/* OMputWCString
 *   If you are using wide char strings you need to output them
 *   with that function rather than with OMputString.
 *   (It takes endianess into account)
```

```
 */
extern OMstatus OMputWCString(OMdev dev, const OMUCS2 * wcstr);
extern OMstatus OMputVar(OMdev dev, const char *var);
extern OMstatus OMputVarN(OMdev dev, const char *var, int len);
extern OMstatus OMputSymbol(OMdev dev, const char *cd, const char *name);
extern OMstatus OMputSymbolN(OMdev dev, const char *cd, int clen,
                             const char *name, int nlen);
extern OMstatus OMputApp(OMdev dev);
extern OMstatus OMputEndApp(OMdev dev);
extern OMstatus OMputAttr(OMdev dev);
extern OMstatus OMputEndAttr(OMdev dev);
extern OMstatus OMputAtp(OMdev dev);
extern OMstatus OMputEndAtp(OMdev dev);
extern OMstatus OMputBind(OMdev dev);
extern OMstatus OMputEndBind(OMdev dev);
extern OMstatus OMputBVar(OMdev dev);
extern OMstatus OMputEndBVar(OMdev dev);
extern OMstatus OMputObject(OMdev dev);
extern OMstatus OMputEndObject(OMdev dev);
extern OMstatus OMputError(OMdev dev);
extern OMstatus OMputEndError(OMdev dev);
extern OMstatus OMputComment(OMdev dev, const char *comment);
extern OMstatus OMputCommentN(OMdev dev, const char *comment, int len);
/* OMgetType
 *   Get the type of the current token on device <dev>/
 * dev: device to look at.
 * type: where to store returned type.
 * return: 0 or some error code
 */
extern OMstatus OMgetType(OMdev dev, OMtokenType * type);
/* OMgetLength
 *   Get the current token length.
 * dev: device to read from
 * len: where to put the token length
 *      the last '\0' for string like tokens is not counted
 *      (rem: for WCString it is the number of bytes not the number of
 *       wide chars)
 * return: 0 or some error code
 */
extern OMstatus OMgetLength(OMdev dev, int *len);
/* OMgetSymbolLength
 *   Get the current token (wich is assumed to be a symbol) lengths.
 * dev: device to read from
 * clen: where to put the cd length (not counting the last '\0')
 * nlen: where to put the name length (not counting the last '\0')
 * return: 0 or some error code
 */
extern OMstatus OMgetSymbolLength(OMdev dev, int *clen, int *nlen);
/* OMGet* functions.
 *    They all take a device <dev> to get token from.
```

```
 *    Some of them need more parameters to fill with the token content.
 *    They are thoroughly documented in OpenMath Specification shiped with
 *    the library.
 * return: a status that reflect the operation success.
 */
extern OMstatus OMgetInt32(OMdev dev, int *i);
extern OMstatus OMgetFloat64(OMdev dev, double *d);
extern OMstatus OMgetBigInt(OMdev dev, char **data, int *len, int *sign,
                            OMbigIntType * format);
extern OMstatus OMgetBigIntN(OMdev dev, char *data, int len, int *sign,
                             OMbigIntType * format);
extern OMstatus OMgetByteArray(OMdev dev, char **data, int *len);
extern OMstatus OMgetByteArrayN(OMdev dev, char *data, int len);
/* OMgetString*
 *   Beware! You are not suposed to use these functions unless you know
 *   for sure you are reading plain 8bits strings.
 *   Thus it is here only for speed/space consideration in very
 *   specific applications.
 *   If input is a 16 bit char string and you read it with these
 *   functions you will lose the 8 most significant bits of each char.
 *   You should rather refer to OMgetWCString* functions.
 */
extern OMstatus OMgetString(OMdev dev, char **str);
extern OMstatus OMgetStringN(OMdev dev, char *str, int len);
/* OMgetWCString*
 *   These functions return 16 bits wide strings. (regardless input
 *   was done in 8 or 16 bits mode).
 *   Thus, most if not all applications should use these functions
 *   preferably to OMgetString*.
 */
extern OMstatus OMgetWCString(OMdev dev, OMUCS2 ** wcstr);
/* BEWARE: the <len> is suposed to be the length in bytes for the
 * preallocated buffer <wcstr> (not the length in number of wide chars)
 */
extern OMstatus OMgetWCStringN(OMdev dev, OMUCS2 * wcstr, int len);
extern OMstatus OMgetVar(OMdev dev, char **var);
extern OMstatus OMgetVarN(OMdev dev, char *var, int len);
extern OMstatus OMgetSymbol(OMdev dev, char **cd, char **name);
extern OMstatus OMgetSymbolN(OMdev dev, char *cd, int clen, char *name,
                             int nlen);
extern OMstatus OMgetApp(OMdev dev);
extern OMstatus OMgetEndApp(OMdev dev);
extern OMstatus OMgetAttr(OMdev dev);
extern OMstatus OMgetEndAttr(OMdev dev);
extern OMstatus OMgetAtp(OMdev dev);
extern OMstatus OMgetEndAtp(OMdev dev);
extern OMstatus OMgetBind(OMdev dev);
extern OMstatus OMgetEndBind(OMdev dev);
extern OMstatus OMgetBVar(OMdev dev);
extern OMstatus OMgetEndBVar(OMdev dev);
```

```
extern OMstatus OMgetObject(OMdev dev);
extern OMstatus OMgetEndObject(OMdev dev);
extern OMstatus OMgetError(OMdev dev);
extern OMstatus OMgetEndError(OMdev dev);
extern OMstatus OMgetComment(OMdev dev, char **comment);
extern OMstatus OMgetCommentN(OMdev dev, char *comment, int len);
/* OMbeginObject
 *   Must be called before every new OpenMath object put.
 *   (Not before every token!)
 * dev: device where new object is to be put.
 * return: status describing operation success
 */
extern OMstatus OMbeginObject(OMdev dev);
/* OMendObject
 *   Must be called after every OpenMath object put.
 *   (Not after every token!)
 * dev: device where object has been put.
 * return: status describing operation success
 */
extern OMstatus OMendObject(OMdev dev);
/* OMignoreComment
 *   Set behavior of a device concerning comments.
 *   (Comments on an input device may safely be ignored.)
 * dev: device to modify
 * set: If set == OMtrue then device will ignore incoming comments
 *      If set == OMfalse then device will process incoming comments
 *                          like other tokens.
 *      By default comments are ignored.
 *      Whatever is <set> value, output of comments is always done.
 * return: previous value
 */
extern OMbool OMignoreComment(OMdev dev, OMbool set);
/* OMtokenCount
 *   Reports the numbe rof tokens that have been in/output on a device
 * dev: devive to examine
 * inTokenNb: where to store number of input tokens (if not NULL)
 * outTokenNb: where to store number of output tokens (if not NULL)
 */
extern void OMtokenCount(OMdev dev, int *inTokenNb, int *outTokenNb);
/* OMgetDeviceEncoding
 *   Get the currnet encoding used by a device
 * dev: device to examine
 * return: current encoding
 */
extern OMencodingType OMgetDeviceEncoding(OMdev dev);
/* OMsetDeviceEncoding
 *   Set the encoding that will be used on a device
 *   BEWARE: changing encoding on a device that has already been used
 *   for IO is unsafe.
 *   but seting encoding on a new device is safe.
```

```
 *   (in some occasions, it is not easy to know which encoding to
 *    use at device creation)
 * dev: device to modify
 * encoding: encoding to use
 */
extern void OMsetDeviceEncoding(OMdev dev, OMencodingType encoding);
/* OMmakeDevice
 *   Create a device from a low level IO
 *   Warning: "IO" should be a "generated" (new) structure as it contains some
 *   state that is private to the device. It is very dangerous for two devices
 *   to share the same "IO" structure.
 * encoding: encoding scheme used by device
 * IO: low level I/O suport for device
 * return: a newly allocated device
 */
extern OMdev OMmakeDevice(OMencodingType encoding, OMIO IO);
/* OMcloseDevice
 *   Close a device previously created with OMmakeDevice
 *   (embedded IO is closed too)
 * dev: device to close
 */
extern void OMcloseDevice(OMdev dev);
/* OMmakeIOFd
 *   Create a low level IO object from a file descriptor.
 *   (May be used on socket for instance.)
 * fd: file descriptor to wrap into the OpenMath IO object.
 * return: a newly allocated IO object.
 */
extern OMIO OMmakeIOFd(int fd);
/* OMmakeIOFile
 *   Create a low level IO object from a FILE*.
 *   (May be used on stdin for instance.)
 * fd: FILE* to wrap into the OpenMath IO object.
 * return: a newly allocated IO object.
 */
extern OMIO OMmakeIOFile(FILE * f);
/* OMmakeIOString
 *   Create a low level IO object from a string (NUL terminator is not needed).
 *   (May be used for copy/paste for instance.)
 * s: pointer to string to use into the OpenMath IO object.
 *    - In case of input device the string must be NUL terminated.
 *    - In case of output device string may be reallocated
 *      to fit size of outcoming objects.
 * return: a newly allocated IO object.
 */
extern OMIO OMmakeIOString(char **s);
/* OMstatusToString
 *   Convert a status to a human readable string that explain its meaning
 * status: status to explain
 * return: corresponding string
```

```
 */
extern char *OMstatusToString(OMstatus status);
/* OMtokenTypeToString
 *   Convert a tokenType to a human readable string
 * ttype: type to convert
 * return: corresponding string
 */
extern char *OMtokenTypeToString(OMtokenType ttype);
/* OMsetVerbosityLevel
 *   When using API some infos may be loged.
 *   This set the required verbosity level.
 * level: level of verbosity.
 *        0 means nothing is nether printed
 *        1 everything is printed (default)
 *        2,... less verbose
 * return: last verbosity level
 */
extern int OMsetVerbosityLevel(int level);
/* OMsetVerbosityOutput
 *   When using API some infos may be loged.
 *   This set the destination for logs.
 * logFile: where to output logs (default is stderr)
 * return: last output
 */
extern FILE *OMsetVerbosityOutput(FILE * logFile);
/* OMlibDynamicInfo
 *   Gather some informations about lib that can't be statically determined.
 *   Complete them with some relevant static infornation too.
 * return: a newly allocated string
 */
extern char *OMlibDynamicInfo(void);
/*** End Prototypes */
/* end of automaticaly updated part */

#ifdef WIN32
#include "windows.h"

/* OMmakeIOHandle
 *   Create a low level IO object from a widows handle.
 * handle: windows handle to wrap into the OpenMath IO object.
 * return: a newly allocated IO object.
 */
extern OMIO OMmakeIOHandle(HANDLE handle);
extern void OMfreeIOHandle(OMIO io);
#endif

#else /* OM_DEV */
/* The prototypes above are in fact collected from all these .h files */
#include "OMbase.h"
#include "OMdev.h"
```

```
#include "OMdevFd.h"
#include "OMdevFile.h"
#include "OMdevString.h"
#include "OMdevHandle.h"
#include "OMencBin.h"
#include "OMencXml.h"
#include "OMmisc.h"
#include "OMutf7.h"
#endif /* OM_DEV */

OMendPrototypes

#endif /* __OM_h__ */
```

70.15 Axiom OpenMath stub functions

These stub functions will eventually be expanded to handle OpenMath. See the OpenMath-Device domain in Volume 10.3. Note that the argument list for the Spad functions does not always match the argument list specified in the OpenMath specification.

There are 4 known OpenMath encodings which are set up in the OpenMathEncoding domain in Volume 10.3.

- Unknown
- Binary
- XML
- SGML

70.15.1 Axiom specific functions

This is used in OpenMathPackage in Volume 10.4.

```
(read OMdev)              -> LispObject
(listCDs)                 -> List(String)
(listSymbols)             -> List(String)
(supportsCD cd)           -> Boolean
(supportsSymbol cd name)  -> Boolean
```

70.15.2 defun om-Read

Read an OpenMath object from dev.

— **defun om-Read** —

```
(defun om-Read (dev)
 (declare (ignore dev)))
```

———-

70.15.3 defun om-listCDs

Lists all of the CDs supported by Axiom.

— **defun om-listCDs** —

```
(defun om-listCDs ())
```

———-

70.15.4 defun om-listSymbols

Lists all the symbols in CD

— **defun om-listSymbols** —

```
(defun om-listSymbols ())
```

———-

70.15.5 defun om-supportsCD

Return true if Axiom supports this CD.

— **defun om-supportsCD** —

```
(defun om-supportsCD (cd)
 (declare (ignore cd)))
```

———-

70.15.6 defun om-supportsSymbol

— **defun om-supportsSymbol** —

```
(defun om-supportsSymbol (cd name)
 (declare (ignore cd name)))
```

———-

70.15.7 Lisp conversion functions

The lisp conversion functions are:

```
(toDev                 LispObject)                   -> OMdev
(fromDev               OMdev)                        -> LispObject
(toStatus              LispObject)                   -> LispObject
(fromStatus            OMstatus)                     -> LispObject
(toEncodingType        LispObject)                   -> OMencodingType
(fromEncodingType      OMencodingType)               -> LispObject
(toBigNumStr           LispObject)                   -> char *
(fromBigNumStr         char *,int,int, OMbigIntType) -> LispObject
(toConn                LispObject)                   -> OMconn
(fromConn              OMconn)                       -> LispObject
(toCString             LispObject)                   -> char **
(fromCString           char **)                      -> LispObject
(lispStringFromCString LispObject)                   -> LispObject
(cStringFromLispString LispObject)                   -> LispObject
```

70.15.8 defun om-setDevEncoding

This sets the encoding used for reading or writeing OpenMath objects to or from dev to enc.

— **defun om-setDevEncoding** —

```
(defun om-setDevEncoding (dev enc)
 (declare (ignore dev enc)))
```

———-

70.15.9 Device manipulation functions

```
(openFileDev LispObject, ints, ...)              -> LispObject
(openStrDev  LispObject, LispObject, LispObject) -> LispObject
(closeDev    LispObject, LispObject)             -> LispObject
```

70.15.10 defun om-openFileDev

This opens file fname for reading or writing OpenMath objects. The mode can be “r” for read, “w” for write, or ”a” for append.

— **defun om-openFileDev** —

```
(defun om-openFileDev (fname fmode enc)
 (declare (ignore fname fmode enc)))
```

———-

70.15.11 defun om-openStringDev

This opens the string str for reading and writing OpenMath objects in encoding enc.

— **defun om-openStringDev** —

```
(defun om-openStringDev (str enc)
 (declare (ignore str enc)))
```

———-

70.15.12 defun om-closeDev

This closes dev, flushing output if necessary.

— **defun om-closeDev** —

```
(defun om-closeDev (dev)
 (declare (ignore dev)))
```

———-

70.15.13 Connection manipulation functions

These are covered in the OpenMathConnection domain in Volume 10.3.

```
(makeConn      LispObject, LispObject) -> LispObject
(closeConn     LispObject, LispObject) -> LispObject
(getConnInDev  LispObject, LispObject) -> LispObject
(getConnOutDev LispObject, LispObject) -> LispObject
```

70.15.14 defun om-makeConn

— **defun om-makeConn** —

```
(defun om-makeConn (conn)
 (declare (ignore conn)))
```

———-

70.15.15 defun om-closeConn

— **defun om-closeConn** —

```
(defun om-closeConn (conn)
 (declare (ignore conn)))
```

————-

70.15.16 defun om-getConnInDev

— **defun om-getConnInDev** —

```
(defun om-getConnInDev (conn)
 (declare (ignore conn)))
```

————-

70.15.17 defun om-getConnOutDev

— **defun om-getConnOutDev** —

```
(defun om-getConnOutDev (conn)
 (declare (ignore conn)))
```

————-

70.15.18 Client/Server functions

These are covered in the OpenMathConnection domain in Volume 10.3. See OMconn.h

```
(bindTCP    LispObject, LispObject, LispObject) -> LispObject
(connectTCP LispObject, int, ...)               -> LispObject
```

70.15.19 defun om-bindTCP

— **defun om-bindTCP** —

```
(defun om-bindTCP (conn port)
 (declare (ignore conn port)))
```

————-

70.15.20 defun om-connectTCP

— defun om-connectTCP —

```
(defun om-connectTCP (conn host port)
 (declare (ignore conn host port)))
```

———-

70.15.21 Device input/output functions

Most of these functions are in the OpenMathDevice domain in Volume 10.3. The only exception seems to be the om-stringPtrToString and om-stringToStringPtr functions which are called in the domains that export primitives. Currently these are:

- Complex (10.3)
- DoubleFloat (10.3)
- Float (10.3)
- Fraction (10.3)
- Integer (10.3)
- List (10.3)
- SingleInteger (10.3)
- String (10.3)
- Symbol (10.3)
- ExpressionToOpenMath (10.4)
- OpenMathPackage (10.4)

Note that putSymbol2 is not implemented.

```
(getApp           LispObject, LispObject) -> LispObject
(getAtp           LispObject, LispObject) -> LispObject
(getAttr          LispObject, LispObject) -> LispObject
(getBind          LispObject, LispObject) -> LispObject
(getBVar          LispObject, LispObject) -> LispObject
(getByteArray     LispObject, LispObject) -> LispObject
(getEndApp        LispObject, LispObject) -> LispObject
(getEndAtp        LispObject, LispObject) -> LispObject
(getEndAttr       LispObject, LispObject) -> LispObject
```

```
(getEndBind        LispObject, LispObject) -> LispObject
(getEndBVar        LispObject, LispObject) -> LispObject
(getEndError       LispObject, LispObject) -> LispObject
(getEndObject      LispObject, LispObject) -> LispObject
(getError          LispObject, LispObject) -> LispObject
(getFloat          LispObject, LispObject) -> LispObject
(getInt            LispObject, LispObject) -> LispObject
(getObject         LispObject, LispObject) -> LispObject
(getString         LispObject, LispObject) -> LispObject
(getSymbol         LispObject, LispObject) -> LispObject
(getType           LispObject, LispObject) -> LispObject
(getVar            LispObject, LispObject) -> LispObject
(putApp            LispObject, LispObject) -> LispObject
(putAtp            LispObject, LispObject) -> LispObject
(putAttr           LispObject, LispObject) -> LispObject
(putBind           LispObject, LispObject) -> LispObject
(putBVar           LispObject, LispObject) -> LispObject
(putByteArray      LispObject, LispObject, LispObject) -> LispObject
(putEndApp         LispObject, LispObject) -> LispObject
(putEndAtp         LispObject, LispObject) -> LispObject
(putEndAttr        LispObject, LispObject) -> LispObject
(putEndBind        LispObject, LispObject) -> LispObject
(putEndBVar        LispObject, LispObject) -> LispObject
(putEndError       LispObject, LispObject) -> LispObject
(putEndObject      LispObject, LispObject) -> LispObject
(putError          LispObject, LispObject) -> LispObject
(putFloat          LispObject, LispObject, LispObject) -> LispObject
(putInt            LispObject, LispObject, LispObject) -> LispObject
(putObject         LispObject, LispObject) -> LispObject
(putString         LispObject, LispObject, LispObject) -> LispObject
(putSymbol         LispObject, LispObject, LispObject) -> LispObject
(putSymbol2        LispObject, int nargs, ...) -> LispObject
(putVar            LispObject, LispObject, LispObject) -> LispObject
(stringPtrToString LispObject, LispObject) -> LispObject
(stringToStringPtr LispObject, LispObject) -> LispObject
```

70.15.22 defun om-getApp

Reads a begin application token from dev.

— **defun om-getApp** —

```
(defun om-getApp (dev)
 (declare (ignore dev)))
```

———-

70.15.23 defun om-getAtp

Reads a begin attribute pair token from dev.
— **defun om-getAtp** —

```
(defun om-getAtp (dev)
 (declare (ignore dev)))
```

———-

70.15.24 defun om-getAttr

Reads a begin attribute token from dev
— **defun om-getAttr** —

```
(defun om-getAttr (dev)
 (declare (ignore dev)))
```

———-

70.15.25 defun om-getBind

Reads a begin binder token from dev.
— **defun om-getBind** —

```
(defun om-getBind (dev)
 (declare (ignore dev)))
```

———-

70.15.26 defun om-getBVar

Reads a begin bound variable list token from dev.
— **defun om-getBVar** —

```
(defun om-getBVar (dev)
 (declare (ignore dev)))
```

———-

70.15.27 defun om-getByteArray

Reads a byte array from dev.
— **defun om-getByteArray** —

```
(defun om-getByteArray (dev))
```

———-

70.15.28 defun om-getEndApp

Reads an end application token from dev
— **defun om-getEndApp** —

```
(defun om-getEndApp (dev)
 (declare (ignore dev)))
```

———-

70.15.29 defun om-getEndAtp

Reads an end attribute pair token from dev.
— **defun om-getEndAtp** —

```
(defun om-getEndAtp (dev)
 (declare (ignore dev)))
```

———-

70.15.30 defun om-getEndAttr

Reads an end attribute token from dev.
— **defun om-getEndAttr** —

```
(defun om-getEndAttr (dev)
 (declare (ignore dev)))
```

———-

70.15.31 defun om-getEndBind

Reads an end binder token from dev.
— **defun om-getEndBind** —

```
(defun om-getEndBind (dev)
 (declare (ignore dev)))
```

———-

70.15.32 defun om-getEndBVar

Reads an end bound variable list token from dev.
— **defun om-getEndBVar** —

```
(defun om-getEndBVar (dev)
 (declare (ignore dev)))
```

———-

70.15.33 defun om-getEndError

Reads an end error token from dev.
— **defun om-getEndError** —

```
(defun om-getEndError (dev)
 (declare (ignore dev)))
```

———-

70.15.34 defun om-getEndObject

Reads an end object token from dev.
— **defun om-getEndObject** —

```
(defun om-getEndObject (dev)
 (declare (ignore dev)))
```

———-

70.15.35 defun om-getError

Reads a begin error token from dev.

— **defun om-getError** —

```
(defun om-getError (dev)
 (declare (ignore dev)))
```

———-

70.15.36 defun om-getFloat

Reads a float from dev.

— **defun om-getFloat** —

```
(defun om-getFloat (dev)
 (declare (ignore dev)))
```

———-

70.15.37 defun om-getInt

Reads an integer from dev.

— **defun om-getInt** —

```
(defun om-getInt (dev)
 (declare (ignore dev)))
```

———-

70.15.38 defun om-getObject

Reads a begin object token from dev.

— **defun om-getObject** —

```
(defun om-getObject (dev)
 (declare (ignore dev)))
```

———-

70.15.39 defun om-getString

Reads a string from dev.

— **defun om-getString** —

```
(defun om-getString (dev)
 (declare (ignore dev)))
```

———-

70.15.40 defun om-getSymbol

Reads a symbol from dev.

— **defun om-getSymbol** —

```
(defun om-getSymbol (dev)
 (declare (ignore dev)))
```

———-

70.15.41 defun om-getType

Returns the type of the next object on dev.

— **defun om-getType** —

```
(defun om-getType (dev)
 (declare (ignore dev)))
```

———-

70.15.42 defun om-getVar

Reads a variable from dev.

— **defun om-getVar** —

```
(defun om-getVar (dev)
 (declare (ignore dev)))
```

———-

70.15.43 defun om-putApp

Writes a begin application token to dev.
— **defun om-putApp** —

```
(defun om-putApp (dev)
 (declare (ignore dev)))
```

———-

70.15.44 defun om-putAtp

This writea a begin application pair token to dev.
— **defun om-putAtp** —

```
(defun om-putAtp (dev)
 (declare (ignore dev)))
```

———-

70.15.45 defun om-putAttr

This writes a begin attribute token to dev.
— **defun om-putAttr** —

```
(defun om-putAttr (dev)
 (declare (ignore dev)))
```

———-

70.15.46 defun om-putBind

This writes a begin binder token to dev.
— **defun om-putBind** —

```
(defun om-putBind (dev)
 (declare (ignore dev)))
```

———-

70.15.47 defun om-putBVar

This writes a begin bound variable list token to dev.

— **defun om-putBVar** —

```
(defun om-putBVar (dev)
 (declare (ignore dev)))
```

———-

70.15.48 defun om-putByteArray

This writes a byte array to dev.

— **defun om-putByteArray** —

```
(defun om-putByteArray (dev b)
 (declare (ignore dev b)))
```

———-

70.15.49 defun om-putEndApp

This writes an end application token to dev.

— **defun om-putEndApp** —

```
(defun om-putEndApp (dev)
 (declare (ignore dev)))
```

———-

70.15.50 defun om-putEndAtp

This writes an end attribute pair to dev.

— **defun om-putEndAtp** —

```
(defun om-putEndAtp (dev)
 (declare (ignore dev)))
```

———-

70.15.51 defun om-putEndAttr

This writes an end attribute token to dev.
— **defun om-putEndAttr** —

```
(defun om-putEndAttr (dev)
 (declare (ignore dev)))
```

———-

70.15.52 defun om-putEndBind

This writes an end binder token to dev.
— **defun om-putEndBind** —

```
(defun om-putEndBind (dev)
 (declare (ignore dev)))
```

———-

70.15.53 defun om-putEndBVar

This writes and end bound variable list token to dev
— **defun om-putEndBVar** —

```
(defun om-putEndBVar (dev)
 (declare (ignore dev)))
```

———-

70.15.54 defun om-putEndError

This writes an end error token to dev
— **defun om-putEndError** —

```
(defun om-putEndError (dev)
 (declare (ignore dev)))
```

———-

70.15.55 defun om-putEndObject

This writes an end object token to dev.

— **defun om-putEndObject** —

```
(defun om-putEndObject (dev)
 (declare (ignore dev)))
```

———-

70.15.56 defun om-putError

This writes a begin error token to dev.

— **defun om-putError** —

```
(defun om-putError (dev)
 (declare (ignore dev)))
```

———-

70.15.57 defun om-putFloat

This writes the float f to dev.

— **defun om-putFloat** —

```
(defun om-putFloat (dev f)
 (declare (ignore dev f)))
```

———-

70.15.58 defun om-putInt

This writes the integer i to dev

— **defun om-putInt** —

```
(defun om-putInt (dev i)
 (declare (ignore dev i)))
```

———-

70.15.59 defun om-putObject

This writes a begin object token to dev.
— **defun om-putObject** —

```
(defun om-putObject (dev)
 (declare (ignore dev)))
```

———-

70.15.60 defun om-putString

This writes the string s to dev.
— **defun om-putString** —

```
(defun om-putString (dev s)
 (declare (ignore dev s)))
```

———-

70.15.61 defun om-putSymbol

This writes the symbol nm using semantics from cd to dev.
— **defun om-putSymbol** —

```
(defun om-putSymbol (dev cd nm)
 (declare (ignore dev cd nm)))
```

———-

70.15.62 defun om-putVar

This writes the variable v to dev.
— **defun om-putVar** —

```
(defun om-putVar (dev v)
 (declare (ignore dev v)))
```

———-

70.15.63 defun om-stringToStringPtr

This is used in the SingleInteger domain in Volume 10.3. This is supposed to return the string from its address? It would appear to be a nop in lisp.

— **defun om-stringToStringPtr** —

```
(defun om-stringToStringPtr (str)
 (declare (ignore str)))
```

———-

70.15.64 defun om-stringPtrToString

This is used in the SingleInteger domain in Volume 10.3. This is supposed to return the string address from a string? It would appear to be a nop in lisp.

— **defun om-stringPtrToString** —

```
(defun om-stringPtrToString (str)
 (declare (ignore str)))
```

———-

Chapter 71

NRLIB code.lisp support code

71.0.65 defun makeByteWordVec2

— defun makeByteWordVec2 0 —

```
(defun |makeByteWordVec2| (maxelement initialvalue)
  (let ((n (cond ((null initialvalue) 7) ('t maxelement))))
    (make-array (length initialvalue)
      :element-type (list 'mod (1+ n))
      :initial-contents initialvalue)))
```

———-

71.0.66 defmacro spadConstant

— defmacro spadConstant 0 —

```
(defmacro |spadConstant| (dollar n)
 `(spadcall (svref ,dollar (the fixnum ,n))))
```

———-

Chapter 72

Monitoring execution

```
MONITOR

This file contains a set of function for monitoring the execution
of the functions in a file. It constructs a hash table that contains
the function name as the key and monitor-data structures as the value

The technique is to use a :cond parameter on trace to call the
monitor-incr function to incr the count every time a function is called

*monitor-table*                                 HASH TABLE
   is the monitor table containing the hash entries
*monitor-nrlibs*                                LIST of STRING
   list of nrlib filenames that are monitored
*monitor-domains*                               LIST of STRING
   list of domains to monitor-report (default is all exposed domains)
monitor-data                                    STRUCTURE
   is the defstruct name of records in the table
   name is the first field and is the name of the monitored function
   count contains a count of times the function was called
   monitorp is a flag that skips counting if nil, counts otherwise
   sourcefile is the name of the file that contains the source code

 ***** SETUP, SHUTDOWN ****

monitor-inittable ()                            FUNCTION
   creates the hashtable and sets *monitor-table*
   note that it is called every time this file is loaded
monitor-end ()                                  FUNCTION
   unhooks all of the trace hooks

 ***** TRACE, UNTRACE *****
```

```
monitor-add (name &optional sourcefile)        FUNCTION
   sets up the trace and adds the function to the table
monitor-delete (fn)                            FUNCTION
   untraces a function and removes it from the table
monitor-enable (&optional fn)                  FUNCTION
   starts tracing for all (or optionally one) functions that
   are in the table
monitor-disable (&optional fn)                 FUNCTION
   stops tracing for all (or optionally one) functions that
   are in the table

***** COUNTING, RECORDING  *****

monitor-reset (&optional fn)                   FUNCTION
   reset the table count for the table (or optionally, for a function)
monitor-incr (fn)                              FUNCTION
   increments the count information for a function
   it is called by trace to increment the count
monitor-decr (fn)                              FUNCTION
   decrements the count information for a function
monitor-info (fn)                              FUNCTION
   returns the monitor-data structure for a function

***** FILE IO *****

monitor-write (items file)                     FUNCTION
   writes a list of symbols or structures to a file
monitor-file (file)                            FUNCTION
   will read a file, scan for defuns, monitor each defun
   NOTE: monitor-file assumes that the file has been loaded

***** RESULTS *****

monitor-results ()                             FUNCTION
   returns a list of the monitor-data structures
monitor-untested ()                            FUNCTION
   returns a list of files that have zero counts
monitor-tested (&optional delete)              FUNCTION
   returns a list of files that have nonzero counts
   optionally calling monitor-delete on those functions

***** CHECKPOINT/RESTORE *****
monitor-checkpoint (file)                     FUNCTION
   save the *monitor-table* in a loadable form
monitor-restore (file)                        FUNCTION
  restore a checkpointed file so that everything is monitored

***** ALGEBRA *****
monitor-autoload ()                           FUNCTION
  traces autoload of algebra to monitor corresponding source files
```

```
  NOTE: this requires the /spad/int/algebra directory
monitor-dirname (args)                          FUNCTION
  expects a list of 1 libstream (loadvol's arglist) and monitors the source
  this is a function called by monitor-autoload
monitor-nrlib (nrlib)                           FUNCTION
  takes an nrlib name as a string (eg POLY) and returns a list of
  monitor-data structures from that source file
monitor-report ()                               FUNCTION
  generate a report of the monitored activity for domains in
  *monitor-domains*
monitor-spadfile (name)                         FUNCTION
  given a spad file, report all nrlibs it creates
  this adds each nrlib name to *monitor-domains* but does not
  trace the functions from those domains
monitor-percent ()                              FUNCTION
  ratio of (functions executed)/(functions traced)
monitor-apropos (str)                           FUNCTION
  given a string, find all monitored symbols containing the string
  the search is case-insensitive. returns a list of monitor-data items

for example:
 suppose we have a file "/u/daly/testmon.lisp" that contains:
 (defun foo1 () (print 'foo1))
 (defun foo2 () (print 'foo2))
 (defun foo3 () (foo1) (foo2) (print 'foo3))
 (defun foo4 () (print 'foo4))

 an example session is:

 ; FIRST WE LOAD THE FILE (WHICH INITS *monitor-table*)

 >(load "/u/daly/monitor.lisp")
 Loading /u/daly/monitor.lisp
 Finished loading /u/daly/monitor.lisp
 T

 ; SECOND WE LOAD THE TESTMON FILE
 >(load "/u/daly/testmon.lisp")
 T

 ; THIRD WE MONITOR THE FILE
 >(monitor-file "/u/daly/testmon.lisp")
 monitoring "/u/daly/testmon.lisp"
 NIL

 ; FOURTH WE CALL A FUNCTION FROM THE FILE (BUMP ITS COUNT)
 >(foo1)

 FOO1
 FOO1
```

```
; AND ANOTHER FUNCTION (BUMP ITS COUNT)
>(foo2)

FOO2
FOO2

; AND A THIRD FUNCTION THAT CALLS THE OTHER TWO (BUMP ALL THREE)
>(foo3)

FOO1
FOO2
FOO3
FOO3

; CHECK THAT THE RESULTS ARE CORRECT

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 2 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 1 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; STOP COUNTING CALLS TO FOO2

>(monitor-disable 'foo2)
NIL

; INVOKE FOO2 THRU FOO3

>(foo3)

FOO1
FOO2
FOO3
FOO3

; NOTICE THAT FOO1 AND FOO3 WERE BUMPED BUT NOT FOO2
>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
```

```
       "/u/daly/testmon.lisp"))

; TEMPORARILY STOP ALL MONITORING

>(monitor-disable)
NIL

; CHECK THAT NOTHING CHANGES

>(foo3)

FOO1
FOO2
FOO3
FOO3

; NO COUNT HAS CHANGED

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 3 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; MONITOR ONLY CALLS TO FOO1

>(monitor-enable 'foo1)
T

; FOO3 CALLS FOO1

>(foo3)

FOO1
FOO2
FOO3
FOO3

; FOO1 HAS CHANGED BUT NOT FOO2 OR FOO3

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 4 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 2 MONITORP NIL SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 2 MONITORP NIL SOURCEFILE
```

```
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; MONITOR EVERYBODY

>(monitor-enable)
NIL

; CHECK THAT EVERYBODY CHANGES

>(foo3)

FOO1
FOO2
FOO3
FOO3

; EVERYBODY WAS BUMPED

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; WHAT FUNCTIONS WERE TESTED?

>(monitor-tested)
(FOO1 FOO2 FOO3)

; WHAT FUNCTIONS WERE NOT TESTED?

>(monitor-untested)
(FOO4)

; UNTRACE THE WHOLE WORLD, MONITORING CANNOT RESTART

>(monitor-end)
NIL

; CHECK THE RESULTS

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
```

```
 #S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))

; CHECK THAT THE FUNCTIONS STILL WORK

>(foo3)

FOO1
FOO2
FOO3
FOO3

; CHECK THAT MONITORING IS NOT OCCURING

>(monitor-results)
(#S(MONITOR-DATA NAME FOO1 COUNT 5 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO2 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp")
 #S(MONITOR-DATA NAME FOO3 COUNT 3 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
 #S(MONITOR-DATA NAME FOO4 COUNT 0 MONITORP T SOURCEFILE
       "/u/daly/testmon.lisp"))
```

72.0.67 defvar $*monitor-domains*

— **initvars** —

```
(defvar *monitor-domains* nil "a list of domains to report")
```

———-

72.0.68 defvar $*monitor-nrlibs*

— **initvars** —

```
(defvar *monitor-nrlibs* nil "a list of nrlibs that have been traced")
```

———-

72.0.69 defvar $*monitor-table*

— **initvars** —

```
(defvar *monitor-table* nil "a table of all of the monitored data")
```

————-

— **postvars** —

```
(eval-when (eval load)
 (unless *monitor-table* (monitor-inittable)))
```

————-

72.0.70 defstruct $monitor-data

— **initvars** —

```
(defstruct monitor-data name count monitorp sourcefile)
```

————-

72.0.71 defstruct $libstream

— **initvars** —

```
(defstruct libstream  mode dirname (indextable nil)  (indexstream nil))
```

————-

72.0.72 defun Initialize the monitor statistics hashtable

[*monitor-table* p1148]

— **defun monitor-inittable 0** —

```
(defun monitor-inittable ()
 "initialize the monitor statistics hashtable"
 (declare (special *monitor-table*))
 (setq *monitor-table* (make-hash-table)))
```

———-

72.0.73 defun End the monitoring process, we cannot restart

[*monitor-table* p1148]

— **defun monitor-end 0** —

```
(defun monitor-end ()
 "End the monitoring process. we cannot restart"
 (declare (special *monitor-table*))
 (maphash
  #'(lambda (key value)
     (declare (ignore value))
     (eval `(untrace ,key)))
   *monitor-table*))
```

———-

72.0.74 defun Return a list of the monitor-data structures

[*monitor-table* p1148]

— **defun monitor-results 0** —

```
(defun monitor-results ()
 "return a list of the monitor-data structures"
 (let (result)
 (declare (special *monitor-table*))
  (maphash
  #'(lambda (key value)
     (declare (ignore key))
     (push value result))
   *monitor-table*)
  (mapcar #'(lambda (x) (pprint x))
   (sort result #'string-lessp :key #'monitor-data-name))))
```

———-

72.0.75 defun Add a function to be monitored

[monitor-delete p1150]
[make-monitor-data p??]
[*monitor-table* p1148]

— **defun monitor-add 0** —

```
(defun monitor-add (name &optional sourcefile)
 "add a function to be monitored"
 (declare (special *monitor-table*))
 (unless (fboundp name) (load sourcefile))
 (when (gethash name *monitor-table*)
  (monitor-delete name))
 (eval `(trace (,name :cond (progn (monitor-incr ',name) nil))))
 (setf (gethash name *monitor-table*)
  (make-monitor-data
     :name name :count 0 :monitorp t :sourcefile sourcefile)))))
```

———-

72.0.76 defun Remove a function being monitored

[*monitor-table* p1148]

— **defun monitor-delete 0** —

```
(defun monitor-delete (fn)
 "Remove a function being monitored"
 (declare (special *monitor-table*))
 (eval `(untrace ,fn))
 (remhash fn *monitor-table*))
```

———-

72.0.77 defun Enable all (or optionally one) function for monitoring

[*monitor-table* p1148]

— **defun monitor-enable 0** —

```
(defun monitor-enable (&optional fn)
 "enable all (or optionally one) function for monitoring"
 (declare (special *monitor-table*))
 (if fn
```

```
  (progn
   (eval `(trace (,fn :cond (progn (monitor-incr ',fn) nil))))
   (setf (monitor-data-monitorp (gethash fn *monitor-table*)) t))
  (maphash
   #'(lambda (key value)
      (declare (ignore value))
      (eval `(trace (,key :cond (progn (monitor-incr ',key) nil))))
      (setf (monitor-data-monitorp (gethash key *monitor-table*)) t))
   *monitor-table*)))
```

———-

72.0.78 defun Disable all (optionally one) function for monitoring

[*monitor-table* p1148]

— defun monitor-disable 0 —

```
(defun monitor-disable (&optional fn)
 "disable all (optionally one) function for monitoring"
 (declare (special *monitor-table*))
 (if fn
  (progn
   (eval `(untrace ,fn))
   (setf (monitor-data-monitorp (gethash fn *monitor-table*)) nil))
  (maphash
   #'(lambda (key value)
      (declare (ignore value))
      (eval `(untrace ,key))
      (setf (monitor-data-monitorp (gethash key *monitor-table*)) nil))
   *monitor-table*)))
```

———-

72.0.79 defun Reset the table count for the table (or a function)

[*monitor-table* p1148]

— defun monitor-reset 0 —

```
(defun monitor-reset (&optional fn)
 "reset the table count for the table (or a function)"
 (declare (special *monitor-table*))
 (if fn
  (setf (monitor-data-count (gethash fn *monitor-table*)) 0)
```

```
  (maphash
   #'(lambda (key value)
      (declare (ignore value))
      (setf (monitor-data-count (gethash key *monitor-table*)) 0))
   *monitor-table*)))
```

———-

72.0.80 defun Incr the count of fn by 1

[*monitor-table* p1148]

— **defun monitor-incr 0** —

```
(defun monitor-incr (fn)
 "incr the count of fn by 1"
 (let (data)
  (declare (special *monitor-table*))
  (setq data (gethash fn *monitor-table*))
  (if data
   (incf (monitor-data-count data))  ;; change table entry by side-effect
   (warn "~s is monitored but not in table..do (untrace ~s)~%" fn fn))))
```

———-

72.0.81 defun Decr the count of fn by 1

[*monitor-table* p1148]

— **defun monitor-decr 0** —

```
(defun monitor-decr (fn)
 "decr the count of fn by 1"
 (let (data)
  (declare (special *monitor-table*))
  (setq data (gethash fn *monitor-table*))
  (if data
   (decf (monitor-data-count data))  ;; change table entry by side-effect
   (warn "~s is monitored but not in table..do (untrace ~s)~%" fn fn))))
```

———-

72.0.82 defun Return the monitor information for a function

[*monitor-table* p1148]

— **defun monitor-info 0** —

```
(defun monitor-info (fn)
 "return the monitor information for a function"
 (declare (special *monitor-table*))
 (gethash fn *monitor-table*))
```

———-

72.0.83 defun Hang a monitor call on all of the defuns in a file

[done p**??**]
[done p**??**]
[monitor-add p1150]

— **defun monitor-file 0** —

```
(defun monitor-file (file)
 "hang a monitor call on all of the defuns in a file"
 (let (expr (package "BOOT"))
  (format t "monitoring ~s~%" file)
  (with-open-file (in file)
   (catch 'done
    (loop
     (setq expr (read in nil 'done))
     (when (eq expr 'done) (throw 'done nil))
     (if (and (consp expr) (eq (car expr) 'in-package))
      (if (and (consp (second expr)) (eq (first (second expr)) 'quote))
       (setq package (string (second (second expr))))
       (setq package (second expr)))
      (when (and (consp expr) (eq (car expr) 'defun))
       (monitor-add (intern (string (second expr)) package) file))))))))
```

———-

72.0.84 defun Return a list of the functions with zero count fields

[*monitor-table* p1148]

— **defun monitor-untested 0** —

```
(defun monitor-untested ()
 "return a list of the functions with zero count fields"
 (let (result)
  (declare (special *monitor-table*))
  (maphash
   #'(lambda (key value)
      (if (and (monitor-data-monitorp value) (= (monitor-data-count value) 0))
       (push key result)))
    *monitor-table*)
 (sort result #'string-lessp )))
```

———-

72.0.85 defun Return a list of functions with non-zero counts

[monitor-delete p1150]
[*monitor-table*) p**??**]

— **defun monitor-tested 0** —

```
(defun monitor-tested (&optional delete)
 "return a list of functions with non-zero counts, optionally deleting them"
 (let (result)
 (declare (special *monitor-table*))
  (maphash
   #'(lambda (key value)
      (when (and (monitor-data-monitorp value)
                 (> (monitor-data-count value) 0))
       (when delete (monitor-delete key))
       (push key result)))
    *monitor-table*)
 (sort result #'string-lessp)))
```

———-

72.0.86 defun Write out a list of symbols or structures to a file

— **defun monitor-write 0** —

```
(defun monitor-write (items file)
 "write out a list of symbols or structures to a file"
 (with-open-file (out file :direction :output)
  (dolist (item items)
    (if (symbolp item)
```

```
    (format out "~s~%" item)
    (format out "~s~50t~s~100t~s~%"
      (monitor-data-sourcefile item)
      (monitor-data-name item)
      (monitor-data-count item))))))
```

———-

72.0.87 defun Save the *monitor-table* in loadable form

[*monitor-table* p1148]
[*print-package* p??]

— **defun monitor-checkpoint 0** —

```
(defun monitor-checkpoint (file)
 "save the *monitor-table* in loadable form"
 (let ((*print-package* t))
  (declare (special *print-package* *monitor-table*))
  (with-open-file (out file :direction :output)
   (format out "(in-package \"BOOT\")~%")
   (format out "(monitor-inittable)~%")
   (dolist (data (monitor-results))
    (format out "(monitor-add '~s ~s)~%"
     (monitor-data-name data)
     (monitor-data-sourcefile data))
    (format out "(setf (gethash '~s *monitor-table*)
                  (make-monitor-data :name '~s :count ~s :monitorp ~s
                                     :sourcefile ~s))~%"
     (monitor-data-name data)
     (monitor-data-name data)
     (monitor-data-count data)
     (monitor-data-monitorp data)
     (monitor-data-sourcefile data))))))
```

———-

72.0.88 defun restore a checkpointed file

— **defun monitor-restore 0** —

```
(defun monitor-restore (file)
 "restore a checkpointed file"
 (load file))
```

———-

72.0.89 defun Printing help documentation

— defun monitor-help 0 —

```
(defun monitor-help ()
 (format t "~%
;;; MONITOR
;;;
;;; This file contains a set of function for monitoring the execution
;;; of the functions in a file. It constructs a hash table that contains
;;; the function name as the key and monitor-data structures as the value
;;;
;;; The technique is to use a :cond parameter on trace to call the
;;; monitor-incr function to incr the count every time a function is called
;;;
;;; *monitor-table*                                HASH TABLE
;;;    is the monitor table containing the hash entries
;;; *monitor-nrlibs*                               LIST of STRING
;;;    list of nrlib filenames that are monitored
;;; *monitor-domains*                              LIST of STRING
;;;    list of domains to monitor-report (default is all exposed domains)
;;; monitor-data                                   STRUCTURE
;;;    is the defstruct name of records in the table
;;;    name is the first field and is the name of the monitored function
;;;    count contains a count of times the function was called
;;;    monitorp is a flag that skips counting if nil, counts otherwise
;;;    sourcefile is the name of the file that contains the source code
;;;
;;;  ***** SETUP, SHUTDOWN ****
;;;
;;; monitor-inittable ()                           FUNCTION
;;;    creates the hashtable and sets *monitor-table*
;;;    note that it is called every time this file is loaded
;;; monitor-end ()                                 FUNCTION
;;;    unhooks all of the trace hooks
;;;
;;;  ***** TRACE, UNTRACE *****
;;;
;;; monitor-add (name &optional sourcefile)        FUNCTION
;;;    sets up the trace and adds the function to the table
;;; monitor-delete (fn)                            FUNCTION
;;;    untraces a function and removes it from the table
;;; monitor-enable (&optional fn)                  FUNCTION
;;;    starts tracing for all (or optionally one) functions that
;;;    are in the table
;;; monitor-disable (&optional fn)                 FUNCTION
```

```
;;;    stops tracing for all (or optionally one) functions that
;;;    are in the table
;;;
;;; ***** COUNTING, RECORDING  *****
;;;
;;; monitor-reset (&optional fn)                  FUNCTION
;;;    reset the table count for the table (or optionally, for a function)
;;; monitor-incr (fn)                             FUNCTION
;;;    increments the count information for a function
;;;    it is called by trace to increment the count
;;; monitor-decr (fn)                             FUNCTION
;;;    decrements the count information for a function
;;; monitor-info (fn)                             FUNCTION
;;;    returns the monitor-data structure for a function
;;;
;;; ***** FILE IO *****
;;;
;;; monitor-write (items file)                    FUNCTION
;;;    writes a list of symbols or structures to a file
;;; monitor-file (file)                           FUNCTION
;;;    will read a file, scan for defuns, monitor each defun
;;;    NOTE: monitor-file assumes that the file has been loaded
;;;
;;; ***** RESULTS *****
;;;
;;; monitor-results ()                            FUNCTION
;;;    returns a list of the monitor-data structures
;;; monitor-untested ()                           FUNCTION
;;;    returns a list of files that have zero counts
;;; monitor-tested (&optional delete)             FUNCTION
;;;    returns a list of files that have nonzero counts
;;;    optionally calling monitor-delete on those functions
;;;
;;; ***** CHECKPOINT/RESTORE *****
;;;
;;; monitor-checkpoint (file)                     FUNCTION
;;;    save the *monitor-table* in a loadable form
;;; monitor-restore (file)                        FUNCTION
;;;   restore a checkpointed file so that everything is monitored
;;;
;;; ***** ALGEBRA *****
;;;
;;; monitor-autoload ()                           FUNCTION
;;;   traces autoload of algebra to monitor corresponding source files
;;;   NOTE: this requires the /spad/int/algebra directory
;;; monitor-dirname (args)                        FUNCTION
;;;   expects a list of 1 libstream (loadvol's arglist) and monitors the source
;;;   this is a function called by monitor-autoload
;;; monitor-nrlib (nrlib)                         FUNCTION
;;;   takes an nrlib name as a string (eg POLY) and returns a list of
```

```
;;;   monitor-data structures from that source file
;;; monitor-report ()                          FUNCTION
;;;   generate a report of the monitored activity for domains in
;;;   *monitor-domains*
;;; monitor-spadfile (name)                    FUNCTION
;;;   given a spad file, report all nrlibs it creates
;;;   this adds each nrlib name to *monitor-domains* but does not
;;;   trace the functions from those domains
;;; monitor-percent ()                         FUNCTION
;;;   ratio of (functions executed)/(functions traced)
;;; monitor-apropos (str)                      FUNCTION
;;;   given a string, find all monitored symbols containing the string
;;;   the search is case-insensitive. returns a list of monitor-data items
") nil)
```

———-

72.0.90 Monitoring algebra files

72.0.91 defun Monitoring algebra code.lsp files

[*monitor-nrlibs* p1147]

— **defun monitor-dirname 0** —

```
(defun monitor-dirname (args)
  "expects a list of 1 libstream (loadvol's arglist) and monitors the source"
 (let (name)
 (declare (special *monitor-nrlibs*))
  (setq name (libstream-dirname (car args)))
  (setq name (file-namestring name))
  (setq name (concatenate 'string "/spad/int/algebra/" name "/code.lsp"))
  (when (probe-file name)
   (push name *monitor-nrlibs*)
   (monitor-file name))))
```

———-

72.0.92 defun Monitor autoloaded files

— **defun monitor-autoload 0** —

```
(defun monitor-autoload ()
```

```
 "traces autoload of algebra to monitor corresponding source files"
 (trace (vmlisp::loadvol
          :entrycond nil
          :exitcond (progn (monitor-dirname system::arglist) nil))))
```

———-

72.0.93 defun Monitor an nrlib

[*monitor-table* p1148]

— **defun monitor-nrlib 0** —

```
(defun monitor-nrlib (nrlib)
 "takes an nrlib name as a string (eg POLY) and returns a list of
  monitor-data structures from that source file"
 (let (result)
 (declare (special *monitor-table*))
  (maphash
   #'(lambda (k v)
      (declare (ignore k))
      (when (string= nrlib
             (pathname-name (car (last
                (pathname-directory (monitor-data-sourcefile v))))))
       (push v result)))
   *monitor-table*)
 result))
```

———-

72.0.94 defun Given a monitor-data item, extract the nrlib name

— **defun monitor-libname 0** —

```
(defun monitor-libname (item)
  "given a monitor-data item, extract the nrlib name"
  (pathname-name (car (last
   (pathname-directory (monitor-data-sourcefile item))))))
```

———-

72.0.95 defun Is this an exposed algebra function?

— **defun monitor-exposedp 0** —

```
(defun monitor-exposedp (fn)
 "exposed functions have more than 1 semicolon. given a symbol, count them"
   (> (count #\; (symbol-name fn)) 1))
```

———-

72.0.96 defun Monitor exposed domains

TPDHERE: note that the file interp.exposed no longer exists. The exposure information is now in bookvol5. This needs to work off the internal exposure list, not the file.
[done p**??**]
[done p**??**]
[*monitor-domains* p1147]

— **defun monitor-readinterp 0** —

```
(defun monitor-readinterp ()
  "read interp.exposed to initialize *monitor-domains* to exposed domains.
   this is the default action. adding or deleting domains from the list
   will change the report results"
  (let (skip expr name)
   (declare (special *monitor-domains*))
   (setq *monitor-domains* nil)
   (with-open-file (in "/spad/src/algebra/interp.exposed")
    (read-line in)
    (read-line in)
    (read-line in)
    (read-line in)
    (catch 'done
     (loop
      (setq expr (read-line in nil "done"))
      (when (string= expr "done") (throw 'done nil))
      (cond
       ((string= expr "basic") (setq skip nil))
       ((string= expr "categories") (setq skip t))
       ((string= expr "hidden") (setq skip t))
       ((string= expr "defaults") (setq skip nil)))
      (when (and (not skip) (> (length expr) 58))
       (setq name (subseq expr 58 (length expr)))
       (setq name (string-right-trim '(#\space) name))
       (when (> (length name) 0)
        (push name *monitor-domains*))))))))
```

———-

72.0.97 defun Generate a report of the monitored domains

[monitor-readinterp p1160]
[*monitor-domains* p1147]

— **defun monitor-report 0** —

```
(defun monitor-report ()
 "generate a report of the monitored activity for domains in *monitor-domains*"
 (let (nrlibs nonzero total)
 (declare (special *monitor-domains*))
  (unless *monitor-domains* (monitor-readinterp))
  (setq nonzero 0)
  (setq total 0)
  (maphash
   #'(lambda (k v)
      (declare (ignore k))
      (let (nextlib point)
       (when (> (monitor-data-count v) 0) (incf nonzero))
       (incf total)
       (setq nextlib (monitor-libname v))
       (setq point (member nextlib nrlibs :test #'string= :key #'car))
       (if point
         (setf (cdr (first point)) (cons v (cdr (first point))))
         (push (cons nextlib (list v)) nrlibs))))
   *monitor-table*)
  (format t "~d of ~d (~d percent) tested~%" nonzero total
    (round (/ (* 100.0 nonzero) total)))
  (setq nrlibs (sort nrlibs #'string< :key #'car))
  (dolist (pair nrlibs)
   (let ((exposedcount 0) (testcount 0))
    (when (member (car pair) *monitor-domains* :test #'string=)
     (format t "for library ~s~%" (car pair))
     (dolist (item (sort (cdr pair)  #'> :key #'monitor-data-count))
      (when (monitor-exposedp (monitor-data-name item))
       (incf exposedcount)
       (when (> (monitor-data-count item) 0) (incf testcount))
       (format t "~5d ~s~%"
         (monitor-data-count item)
         (monitor-data-name item))))
     (if (= exposedcount testcount)
      (format t "~a has all exposed functions tested~%" (car pair))
      (format t "Daly bug:~a has untested exposed functions~%" (car pair))))))
 nil))
```

———-

72.0.98 defun Parse an)abbrev expression for the domain name

— **defun monitor-parse 0** —

```
(defun monitor-parse (expr)
  (let (point1 point2)
   (setq point1 (position #\space expr :test #'char=))
   (setq point1 (position #\space expr :start point1 :test-not #'char=))
   (setq point1 (position #\space expr :start point1 :test #'char=))
   (setq point1 (position #\space expr :start point1 :test-not #'char=))
   (setq point2 (position #\space expr :start point1 :test #'char=))
   (subseq expr point1 point2)))
```

———-

72.0.99 defun Given a spad file, report all nrlibs it creates

[done p**??**]
[done p**??**]
[monitor-parse p1162]
[*monitor-domains* p1147]

— **defun monitor-spadfile 0** —

```
(defun monitor-spadfile (name)
 "given a spad file, report all nrlibs it creates"
 (let (expr)
 (declare (special *monitor-domains*))
  (with-open-file (in name)
   (catch 'done
    (loop
     (setq expr (read-line in nil 'done))
     (when (eq expr 'done) (throw 'done nil))
     (when (and (> (length expr) 4) (string= (subseq expr 0 4) ")abb"))
      (setq *monitor-domains*
       (adjoin (monitor-parse expr) *monitor-domains* :test #'string=))))))))
```

———-

72.0.100 defun Print percent of functions tested

[*monitor-table* p1148]

— **defun monitor-percent 0** —

```
(defun monitor-percent ()
 "Print percent of functions tested"
 (let (nonzero total)
 (declare (special *monitor-table*))
  (setq nonzero 0)
  (setq total 0)
  (maphash
   #'(lambda (k v)
      (declare (ignore k))
      (when (> (monitor-data-count v) 0) (incf nonzero))
      (incf total))
   *monitor-table*)
   (format t "~d of ~d (~d percent) tested~%" nonzero total
     (round (/ (* 100.0 nonzero) total)))))
```

———-

72.0.101 defun Find all monitored symbols containing the string

[*monitor-table* p1148]

— **defun monitor-apropos 0** —

```
(defun monitor-apropos (str)
  "given a string, find all monitored symbols containing the string
   the search is case-insensitive. returns a list of monitor-data items"
 (let (result)
  (maphash
   #'(lambda (k v)
      (when
       (search (string-upcase str)
               (string-upcase (symbol-name k))
               :test #'string=)
        (push v result)))
   *monitor-table*)
 result))
```

———-

Chapter 73

The Interpreter

— **Interpreter** —

```
(setq *print-array* nil)
(setq *print-circle* nil)
(setq *print-pretty* nil)

(in-package "BOOT")
\getchunk{initvars}

;;; level 0 macros

\getchunk{defmacro bit-to-truth 0}
\getchunk{defmacro bvec-elt 0}
\getchunk{defmacro idChar? 0}
\getchunk{defmacro identp 0}
\getchunk{defmacro qsabsval 0}
\getchunk{defmacro qsadd1 0}
\getchunk{defmacro qsdifference 0}
\getchunk{defmacro qslessp 0}
\getchunk{defmacro qsmax 0}
\getchunk{defmacro qsmin 0}
\getchunk{defmacro qsminus 0}
\getchunk{defmacro qsoddp 0}
\getchunk{defmacro qsplus 0}
\getchunk{defmacro qssub1 0}
\getchunk{defmacro qstimes 0}
\getchunk{defmacro qszerop 0}
\getchunk{defmacro spadConstant 0}

;;; above level 0 macros
```

```
\getchunk{defmacro assq}
\getchunk{defmacro bvec-setelt}
\getchunk{defmacro bvec-size}
\getchunk{defmacro cdaref2}
\getchunk{defmacro cdelt}
\getchunk{defmacro cdlen}
\getchunk{defmacro cdancols}
\getchunk{defmacro cdanrows}
\getchunk{defmacro cdsetaref2}
\getchunk{defmacro cdsetelt}
\getchunk{defmacro danrows}
\getchunk{defmacro dancols}
\getchunk{defmacro daref2}
\getchunk{defmacro delt}
\getchunk{defmacro DFAdd}
\getchunk{defmacro DFAcos}
\getchunk{defmacro DFAcosh}
\getchunk{defmacro DFAsin}
\getchunk{defmacro DFAsinh}
\getchunk{defmacro DFAtan}
\getchunk{defmacro DFAtan2}
\getchunk{defmacro DFAtanh}
\getchunk{defmacro DFCos}
\getchunk{defmacro DFCosh}
\getchunk{defmacro DFDivide}
\getchunk{defmacro DFEql}
\getchunk{defmacro DFExp}
\getchunk{defmacro DFExpt}
\getchunk{defmacro DFIntegerDivide}
\getchunk{defmacro DFIntegerExpt}
\getchunk{defmacro DFIntegerMultiply}
\getchunk{defmacro DFLessThan}
\getchunk{defmacro DFLog}
\getchunk{defmacro DFLogE}
\getchunk{defmacro DFMax}
\getchunk{defmacro DFMin}
\getchunk{defmacro DFMinusp}
\getchunk{defmacro DFMultiply}
\getchunk{defmacro DFSin}
\getchunk{defmacro DFSinh}
\getchunk{defmacro DFSqrt}
\getchunk{defmacro DFSubtract}
\getchunk{defmacro DFTan}
\getchunk{defmacro DFTanh}
\getchunk{defmacro DFUnaryMinus}
\getchunk{defmacro DFZerop}
\getchunk{defmacro dlen}
\getchunk{defmacro dsetaref2}
\getchunk{defmacro dsetelt}
\getchunk{defmacro elt32}
```

```
\getchunk{defmacro funfind}
\getchunk{defmacro hget}
\getchunk{defmacro make-cdouble-matrix}
\getchunk{defmacro make-cdouble-vector}
\getchunk{defmacro make-double-matrix}
\getchunk{defmacro make-double-matrix1}
\getchunk{defmacro make-double-vector}
\getchunk{defmacro make-double-vector1}
\getchunk{defmacro qv32len}
\getchunk{defmacro Rest}
\getchunk{defmacro startsId?}
\getchunk{defmacro setelt32}
\getchunk{defmacro trapNumericErrors}
\getchunk{defmacro truth-to-bit}
\getchunk{defmacro while}
\getchunk{defmacro whileWithResult}

;;; layer 0 (all common lisp)

\getchunk{defun acot 0}
\getchunk{defun acoth 0}
\getchunk{defun acsc 0}
\getchunk{defun acsch 0}
\getchunk{defun asec 0}
\getchunk{defun asech 0}
\getchunk{defun axiomVersion 0}

\getchunk{defun BooleanEquality 0}
\getchunk{defun bvec-and 0}
\getchunk{defun bvec-concat 0}
\getchunk{defun bvec-copy 0}
\getchunk{defun bvec-equal 0}
\getchunk{defun bvec-greater 0}
\getchunk{defun bvec-make-full 0}
\getchunk{defun bvec-nand 0}
\getchunk{defun bvec-nor 0}
\getchunk{defun bvec-not 0}
\getchunk{defun bvec-or 0}
\getchunk{defun bvec-xor 0}

\getchunk{defun cleanupLine 0}
\getchunk{defun clearMacroTable 0}
\getchunk{defun concat 0}
\getchunk{defun cot 0}
\getchunk{defun coth 0}
\getchunk{defun createCurrentInterpreterFrame 0}
\getchunk{defun credits 0}
\getchunk{defun csc 0}
\getchunk{defun csch 0}
```

```
\getchunk{defun Delay 0}
\getchunk{defun desiredMsg 0}
\getchunk{defun DirToString 0}
\getchunk{defun divide2 0}
\getchunk{defun dqAppend 0}
\getchunk{defun dqToList 0}
\getchunk{defun dqUnit 0}

\getchunk{defun emptyInterpreterFrame 0}

\getchunk{defun fin 0}
\getchunk{defun findFrameInRing 0}
\getchunk{defun flatten 0}
\getchunk{defun fnameExists? 0}
\getchunk{defun fnameName 0}
\getchunk{defun fnameReadable? 0}
\getchunk{defun fnameType 0}
\getchunk{defun frameExposureData 0}
\getchunk{defun frameHiFiAccess 0}
\getchunk{defun frameHistList 0}
\getchunk{defun frameHistListAct 0}
\getchunk{defun frameHistListLen 0}
\getchunk{defun frameHistoryTable 0}
\getchunk{defun frameHistRecord 0}
\getchunk{defun frameInteractive 0}
\getchunk{defun frameIOIndex 0}
\getchunk{defun frameName 0}
\getchunk{defun frameNames 0}
\getchunk{defun From 0}
\getchunk{defun FromTo 0}

\getchunk{defun get-current-directory 0}
\getchunk{defun getenviron 0}
\getchunk{defun getLinePos 0}
\getchunk{defun getLineText 0}
\getchunk{defun getMsgArgL 0}
\getchunk{defun getMsgKey 0}
\getchunk{defun getMsgKey? 0}
\getchunk{defun getMsgPrefix 0}
\getchunk{defun getMsgPosTagOb 0}
\getchunk{defun getMsgPrefix? 0}
\getchunk{defun getMsgTag 0}
\getchunk{defun getMsgTag? 0}
\getchunk{defun getMsgText 0}
\getchunk{defun getParserMacroNames 0}
\getchunk{defun getPreStL 0}

\getchunk{defun hasOptArgs? 0}

\getchunk{defun incActive? 0}
```

```
\getchunk{defun incCommand? 0}
\getchunk{defun incDrop 0}
\getchunk{defun incHandleMessage 0}
\getchunk{defun inclmsgConsole 0}
\getchunk{defun inclmsgFinSkipped 0}
\getchunk{defun inclmsgPrematureEOF 0}
\getchunk{defun inclmsgCmdBug 0}
\getchunk{defun inclmsgIfBug 0}
\getchunk{defun incPrefix? 0}
\getchunk{defun init-memory-config 0}
\getchunk{defun insertPos 0}
\getchunk{defun integer-decode-float-denominator 0}
\getchunk{defun integer-decode-float-exponent 0}
\getchunk{defun integer-decode-float-sign 0}
\getchunk{defun integer-decode-float-numerator 0}
\getchunk{defun intloopPrefix? 0}
\getchunk{defun isIntegerString 0}

\getchunk{defun keyword 0}
\getchunk{defun keyword? 0}

\getchunk{defun lfcomment 0}
\getchunk{defun lferror 0}
\getchunk{defun lffloat 0}
\getchunk{defun lfid 0}
\getchunk{defun lfinteger 0}
\getchunk{defun lfnegcomment 0}
\getchunk{defun lfrinteger 0}
\getchunk{defun lfspaces 0}
\getchunk{defun lfstring 0}
\getchunk{defun lnCreate 0}
\getchunk{defun lnExtraBlanks 0}
\getchunk{defun lnFileName? 0}
\getchunk{defun lnGlobalNum 0}
\getchunk{defun lnImmediate? 0}
\getchunk{defun lnLocalNum 0}
\getchunk{defun lnPlaceOfOrigin 0}
\getchunk{defun lnSetGlobalNum 0}
\getchunk{defun lnString 0}

\getchunk{defun macODefine 0}
\getchunk{defun macOInfiniteExpansion,name 0}
\getchunk{defun make-absolute-filename 0}
\getchunk{defun makeByteWordVec2 0}
\getchunk{defun makeInitialModemapFrame 0}
\getchunk{defun manexp 0}
\getchunk{defun member 0}
\getchunk{defun monitor-add 0}
\getchunk{defun monitor-apropos 0}
\getchunk{defun monitor-autoload 0}
```

```
\getchunk{defun monitor-checkpoint 0}
\getchunk{defun monitor-decr 0}
\getchunk{defun monitor-delete 0}
\getchunk{defun monitor-dirname 0}
\getchunk{defun monitor-disable 0}
\getchunk{defun monitor-enable 0}
\getchunk{defun monitor-end 0}
\getchunk{defun monitor-exposedp 0}
\getchunk{defun monitor-file 0}
\getchunk{defun monitor-help 0}
\getchunk{defun monitor-incr 0}
\getchunk{defun monitor-info 0}
\getchunk{defun monitor-inittable 0}
\getchunk{defun monitor-libname 0}
\getchunk{defun monitor-nrlib 0}
\getchunk{defun monitor-parse 0}
\getchunk{defun monitor-percent 0}
\getchunk{defun monitor-readinterp 0}
\getchunk{defun monitor-report 0}
\getchunk{defun monitor-reset 0}
\getchunk{defun monitor-restore 0}
\getchunk{defun monitor-results 0}
\getchunk{defun monitor-spadfile 0}
\getchunk{defun monitor-tested 0}
\getchunk{defun monitor-untested 0}
\getchunk{defun monitor-write 0}

\getchunk{defun ncError 0}
\getchunk{defun ncloopEscaped 0}
\getchunk{defun ncloopPrefix? 0}
\getchunk{defun ncloopPrintLines 0}
\getchunk{defun nonBlank 0}
\getchunk{defun npAnyNo 0}
\getchunk{defun npboot 0}
\getchunk{defun npEqPeek 0}
\getchunk{defun nplisp 0}
\getchunk{defun npPop1 0}
\getchunk{defun npPop2 0}
\getchunk{defun npPop3 0}
\getchunk{defun npPush 0}

\getchunk{defun opTran 0}

\getchunk{defun pfAndLeft 0}
\getchunk{defun pfAndRight 0}
\getchunk{defun pfAppend 0}
\getchunk{defun pfApplicationArg 0}
\getchunk{defun pfApplicationOp 0}
\getchunk{defun pfAssignLhsItems 0}
\getchunk{defun pf0AssignLhsItems 0}
```

```
\getchunk{defun pfAssignRhs 0}
\getchunk{defun pfBreakFrom 0}
\getchunk{defun pfCoercetoExpr 0}
\getchunk{defun pfCoercetoType 0}
\getchunk{defun pfCollectBody 0}
\getchunk{defun pfCollectIterators 0}
\getchunk{defun pfDefinitionLhsItems 0}
\getchunk{defun pfDefinitionRhs 0}
\getchunk{defun pfDoBody 0}
\getchunk{defun pfExitCond 0}
\getchunk{defun pfExitExpr 0}
\getchunk{defun pfFirst 0}
\getchunk{defun pfFreeItems 0}
\getchunk{defun pfForinLhs 0}
\getchunk{defun pfForinWhole 0}
\getchunk{defun pfFromdomDomain 0}
\getchunk{defun pfFromdomWhat 0}
\getchunk{defun pfIfCond 0}
\getchunk{defun pfIfElse 0}
\getchunk{defun pfIfThen 0}
\getchunk{defun pfLambdaArgs 0}
\getchunk{defun pfLambdaBody 0}
\getchunk{defun pfLambdaRets 0}
\getchunk{defun pfLiteral? 0}
\getchunk{defun pfLocalItems 0}
\getchunk{defun pfLoopIterators 0}
\getchunk{defun pfMacroLhs 0}
\getchunk{defun pfMacroRhs 0}
\getchunk{defun pfMLambdaArgs 0}
\getchunk{defun pfMLambdaBody 0}
\getchunk{defun pfNotArg 0}
\getchunk{defun pfNovalueExpr 0}
\getchunk{defun pfOrLeft 0}
\getchunk{defun pfOrRight 0}
\getchunk{defun pfParts 0}
\getchunk{defun pfPile 0}
\getchunk{defun pfPretendExpr 0}
\getchunk{defun pfPretendType 0}
\getchunk{defun pfRestrictExpr 0}
\getchunk{defun pfRestrictType 0}
\getchunk{defun pfReturnExpr 0}
\getchunk{defun pfRuleLhsItems 0}
\getchunk{defun pfRuleRhs 0}
\getchunk{defun pfSecond 0}
\getchunk{defun pfSequenceArgs 0}
\getchunk{defun pfSuchthatCond 0}
\getchunk{defun pfTaggedExpr 0}
\getchunk{defun pfTaggedTag 0}
\getchunk{defun pfTree 0}
\getchunk{defun pfTypedId 0}
```

```
\getchunk{defun pfTypedType 0}
\getchunk{defun pfTupleParts 0}
\getchunk{defun pfWhereContext 0}
\getchunk{defun pfWhereExpr 0}
\getchunk{defun pfWhileCond 0}
\getchunk{defun pmDontQuote? 0}
\getchunk{defun poCharPosn 0}
\getchunk{defun poGetLineObject 0}
\getchunk{defun poNopos? 0}
\getchunk{defun poNoPosition 0}
\getchunk{defun poNoPosition? 0}
\getchunk{defun printAsTeX 0}
\getchunk{defun pname 0}

\getchunk{defun qenum 0}
\getchunk{defun qsquotient 0}
\getchunk{defun qsremainder 0}
\getchunk{defun quotient2 0}

\getchunk{defun random 0}
\getchunk{defun rdigit? 0}
\getchunk{defun reclaim 0}
\getchunk{defun remainder2 0}
\getchunk{defun remLine 0}
\getchunk{defun rep 0}
\getchunk{defun resetStackLimits 0}

\getchunk{defun sameUnionBranch 0}
\getchunk{defun satisfiesUserLevel 0}
\getchunk{defun scanCloser? 0}
\getchunk{defun sec 0}
\getchunk{defun sech 0}
\getchunk{defun setCurrentLine 0}
\getchunk{defun setMsgPrefix 0}
\getchunk{defun setMsgText 0}
\getchunk{defun set-restart-hook 0}
\getchunk{defun showMsgPos? 0}
\getchunk{defun StreamNull 0}
\getchunk{defun stripLisp 0}
\getchunk{defun stripSpaces 0}

\getchunk{defun theid 0}
\getchunk{defun thefname 0}
\getchunk{defun theorigin 0}
\getchunk{defun tokPart 0}
\getchunk{defun To 0}
\getchunk{defun Top? 0}
\getchunk{defun trademark 0}

\getchunk{defun zeroOneTran 0}
```

```
;;; above level 0

\getchunk{defun abbQuery}
\getchunk{defun abbreviations}
\getchunk{defun abbreviationsSpad2Cmd}
\getchunk{defun addBinding}
\getchunk{defun addBindingInteractive}
\getchunk{defun addInputLibrary}
\getchunk{defun addNewInterpreterFrame}
\getchunk{defun addoperations}
\getchunk{defun addTraceItem}
\getchunk{defun algCoerceInteractive}
\getchunk{defun allConstructors}
\getchunk{defun allOperations}
\getchunk{defun alqlGetOrigin}
\getchunk{defun alqlGetParams}
\getchunk{defun alqlGetKindString}
\getchunk{defun alreadyOpened?}
\getchunk{defun apropos}
\getchunk{defun assertCond}
\getchunk{defun augmentTraceNames}

\getchunk{defun basicLookup}
\getchunk{defun basicLookupCheckDefaults}
\getchunk{defun break}
\getchunk{defun breaklet}
\getchunk{defun brightprint}
\getchunk{defun brightprint-0}
\getchunk{defun browse}
\getchunk{defun browseOpen}

\getchunk{defun cacheKeyedMsg}
\getchunk{defun categoryOpen}
\getchunk{defun changeHistListLen}
\getchunk{defun changeToNamedInterpreterFrame}
\getchunk{defun charDigitVal}
\getchunk{defun cleanline}
\getchunk{defun clear}
\getchunk{defun clearCmdAll}
\getchunk{defun clearCmdCompletely}
\getchunk{defun clearCmdExcept}
\getchunk{defun clearCmdParts}
\getchunk{defun clearCmdSortedCaches}
\getchunk{defun clearFrame}
\getchunk{defun clearParserMacro}
\getchunk{defun clearSpad2Cmd}
\getchunk{defun close}
\getchunk{defun closeInterpreterFrame}
\getchunk{defun cmpnote}
```

```
\getchunk{defun coerceSpadArgs2E}
\getchunk{defun coerceSpadFunValue2E}
\getchunk{defun coerceTraceArgs2E}
\getchunk{defun coerceTraceFunValue2E}
\getchunk{defun commandAmbiguityError}
\getchunk{defun commandError}
\getchunk{defun commandErrorIfAmbiguous}
\getchunk{defun commandErrorMessage}
\getchunk{defun commandsForUserLevel}
\getchunk{defun commandUserLevelError}
\getchunk{defun compareposns}
\getchunk{defun compileBoot}
\getchunk{defun compiledLookup}
\getchunk{defun compiledLookupCheck}
\getchunk{defun compressOpen}
\getchunk{defun constoken}
\getchunk{defun constructSubst}
\getchunk{defun containsVars}
\getchunk{defun containsVars1}
\getchunk{defun copyright}
\getchunk{defun countCache}

\getchunk{defun DaaseName}
\getchunk{defun decideHowMuch}
\getchunk{defun defaultTargetFE}
\getchunk{defun defiostream}
\getchunk{defun deldatabase}
\getchunk{defun deleteFile}
\getchunk{defun describe}
\getchunk{defun describeFortPersistence}
\getchunk{defun describeInputLibraryArgs}
\getchunk{defun describeOutputLibraryArgs}
\getchunk{defun describeSetFortDir}
\getchunk{defun describeSetFortTmpDir}
\getchunk{defun describeSetFunctionsCache}
\getchunk{defun describeSetLinkerArgs}
\getchunk{defun describeSetNagHost}
\getchunk{defun describeSetOutputAlgebra}
\getchunk{defun describeSetOutputFormula}
\getchunk{defun describeSetOutputFortran}
\getchunk{defun describeSetOutputHtml}
\getchunk{defun describeSetOutputMathml}
\getchunk{defun describeSetOutputOpenMath}
\getchunk{defun describeSetOutputTex}
\getchunk{defun describeSetStreamsCalculate}
\getchunk{defun describeSpad2Cmd}
\getchunk{defun dewritify}
\getchunk{defun dewritify,dewritifyInner}
\getchunk{defun diffAlist}
\getchunk{defun digit?}
```

```
\getchunk{defun digitp}
\getchunk{defun disableHist}
\getchunk{defun display}
\getchunk{defun displayCondition}
\getchunk{defun displayExposedConstructors}
\getchunk{defun displayExposedGroups}
\getchunk{defun displayFrameNames}
\getchunk{defun displayHiddenConstructors}
\getchunk{defun displayMacro}
\getchunk{defun displayMacros}
\getchunk{defun displayMode}
\getchunk{defun displayModemap}
\getchunk{defun displayOperations}
\getchunk{defun displayOperationsFromLisplib}
\getchunk{defun displayParserMacro}
\getchunk{defun displayProperties}
\getchunk{defun displayProperties,sayFunctionDeps}
\getchunk{defun displaySetOptionInformation}
\getchunk{defun displaySetVariableSettings}
\getchunk{defun displaySpad2Cmd}
\getchunk{defun displayType}
\getchunk{defun displayValue}
\getchunk{defun displayWorkspaceNames}
\getchunk{defun domainToGenvar}
\getchunk{defun domArg}
\getchunk{defun domArg2}
\getchunk{defun doSystemCommand}
\getchunk{defun dqConcat}
\getchunk{defun dropInputLibrary}
\getchunk{defun dumbTokenize}

\getchunk{defun edit}
\getchunk{defun editFile}
\getchunk{defun editSpad2Cmd}
\getchunk{defun Else?}
\getchunk{defun Elseif?}
\getchunk{defun enPile}
\getchunk{defun eofp}
\getchunk{defun eqpileTree}
\getchunk{defun erMsgCompare}
\getchunk{defun erMsgSep}
\getchunk{defun erMsgSort}
\getchunk{defun evalCategory}
\getchunk{defun evalDomain}
\getchunk{defun evaluateSignature}
\getchunk{defun evaluateType}
\getchunk{defun evaluateType1}
\getchunk{defun ExecuteInterpSystemCommand}
\getchunk{defun executeQuietCommand}
```

```
\getchunk{defun fetchKeyedMsg}
\getchunk{defun fetchOutput}
\getchunk{defun fillerSpaces}
\getchunk{defun filterAndFormatConstructors}
\getchunk{defun filterListOfStrings}
\getchunk{defun filterListOfStringsWithFn}
\getchunk{defun firstTokPosn}
\getchunk{defun fixObjectForPrinting}
\getchunk{defun flattenOperationAlist}
\getchunk{defun float2Sex}
\getchunk{defun fnameDirectory}
\getchunk{defun fnameMake}
\getchunk{defun fnameNew}
\getchunk{defun fnameWritable?}
\getchunk{defun frame}
\getchunk{defun frameEnvironment}
\getchunk{defun frameSpad2Cmd}
\getchunk{defun functionp}
\getchunk{defun funfind,LAM}

\getchunk{defun genDomainTraceName}
\getchunk{defun gensymInt}
\getchunk{defun getAliasIfTracedMapParameter}
\getchunk{defun getAndEvalConstructorArgument}
\getchunk{defun getAndSay}
\getchunk{defun getBpiNameIfTracedMap}
\getchunk{defun getBrowseDatabase}
\getchunk{defun getdatabase}
\getchunk{defun getDirectoryList}
\getchunk{defun getFirstWord}
\getchunk{defun getKeyedMsg}
\getchunk{defun getMapSig}
\getchunk{defun getMapSubNames}
\getchunk{defun getMsgCatAttr}
\getchunk{defun getMsgFTTag?}
\getchunk{defun getMsgInfoFromKey}
\getchunk{defun getMsgLitSym}
\getchunk{defun getMsgPos}
\getchunk{defun getMsgPos2}
\getchunk{defun getMsgToWhere}
\getchunk{defun getOplistForConstructorForm}
\getchunk{defun getOplistWithUniqueSignatures}
\getchunk{defun getOption}
\getchunk{defun getPosStL}
\getchunk{defun getPreviousMapSubNames}
\getchunk{defun getProplist}
\getchunk{defun getrefv32}
\getchunk{defun getStFromMsg}
\getchunk{defun getSystemCommandLine}
\getchunk{defun getTraceOption}
```

```
\getchunk{defun getTraceOption,hn}
\getchunk{defun getTraceOptions}
\getchunk{defun getWorkspaceNames}

\getchunk{defun handleNoParseCommands}
\getchunk{defun handleParsedSystemCommands}
\getchunk{defun handleTokensizeSystemCommands}
\getchunk{defun hasAtt}
\getchunk{defun hasAttSig}
\getchunk{defun hasCatExpression}
\getchunk{defun hasCate}
\getchunk{defun hasCateSpecial}
\getchunk{defun hasCateSpecialNew}
\getchunk{defun hasCate1}
\getchunk{defun hasCaty}
\getchunk{defun hasCaty1}
\getchunk{defun hashable}
\getchunk{defun hasOption}
\getchunk{defun hasPair}
\getchunk{defun hasSig}
\getchunk{defun hasSigAnd}
\getchunk{defun hasSigOr}
\getchunk{defun help}
\getchunk{defun helpSpad2Cmd}
\getchunk{defun histFileErase}
\getchunk{defun histFileName}
\getchunk{defun histInputFileName}
\getchunk{defun history}
\getchunk{defun historySpad2Cmd}
\getchunk{defun hkeys}
\getchunk{defun hput}

\getchunk{defun If?}
\getchunk{defun ifCond}
\getchunk{defun importFromFrame}
\getchunk{defun incAppend}
\getchunk{defun incAppend1}
\getchunk{defun incBiteOff}
\getchunk{defun incClassify}
\getchunk{defun incCommandTail}
\getchunk{defun incConsoleInput}
\getchunk{defun incFileInput}
\getchunk{defun incFileName}
\getchunk{defun incIgen}
\getchunk{defun incIgen1}
\getchunk{defun inclFname}
\getchunk{defun incLine}
\getchunk{defun incLine1}
\getchunk{defun inclmsgCannotRead}
\getchunk{defun inclmsgFileCycle}
```

```
\getchunk{defun inclmsgPrematureFin}
\getchunk{defun incLude}
\getchunk{defun incLude1}
\getchunk{defun inclmsgConActive}
\getchunk{defun inclmsgConStill}
\getchunk{defun inclmsgIfSyntax}
\getchunk{defun inclmsgNoSuchFile}
\getchunk{defun inclmsgSay}
\getchunk{defun incNConsoles}
\getchunk{defun incRenumber}
\getchunk{defun incRenumberItem}
\getchunk{defun incRenumberLine}
\getchunk{defun incRgen}
\getchunk{defun incRgen1}
\getchunk{defun incStream}
\getchunk{defun incString}
\getchunk{defun incZip}
\getchunk{defun incZip1}
\getchunk{defun init-boot/spad-reader}
\getchunk{defun initHist}
\getchunk{defun initHistList}
\getchunk{defun initial-getdatabase}
\getchunk{defun initializeInterpreterFrameRing}
\getchunk{defun initializeSetVariables}
\getchunk{defun initImPr}
\getchunk{defun initroot}
\getchunk{defun initToWhere}
\getchunk{defun insertAlist}
\getchunk{defun insertpile}
\getchunk{defun InterpExecuteSpadSystemCommand}
\getchunk{defun interpFunctionDepAlists}
\getchunk{defun interpOpen}
\getchunk{defun interpret}
\getchunk{defun interpret1}
\getchunk{defun interpret2}
\getchunk{defun interpretTopLevel}
\getchunk{defun intInterpretPform}
\getchunk{defun intloop}
\getchunk{defun intloopEchoParse}
\getchunk{defun intloopInclude}
\getchunk{defun intloopInclude0}
\getchunk{defun intnplisp}
\getchunk{defun intloopProcess}
\getchunk{defun intloopProcessString}
\getchunk{defun intloopReadConsole}
\getchunk{defun intloopSpadProcess}
\getchunk{defun intloopSpadProcess,interp}
\getchunk{defun intProcessSynonyms}
\getchunk{defun ioclear}
\getchunk{defun iostat}
```

```
\getchunk{defun isDomainOrPackage}
\getchunk{defun isDomainValuedVariable}
\getchunk{defun isEqualOrSubDomain}
\getchunk{defun isExposedConstructor}
\getchunk{defun isgenvar}
\getchunk{defun isInterpOnlyMap}
\getchunk{defun isListOfIdentifiers}
\getchunk{defun isListOfIdentifiersOrStrings}
\getchunk{defun isPartialMode}
\getchunk{defun isSharpVar}
\getchunk{defun isSharpVarWithNum}
\getchunk{defun isSubForRedundantMapName}
\getchunk{defun isSystemDirectory}
\getchunk{defun isTraceGensym}
\getchunk{defun isUncompiledMap}

\getchunk{defun justifyMyType}

\getchunk{defun KeepPart?}

\getchunk{defun lassocSub}
\getchunk{defun lastTokPosn}
\getchunk{defun leader?}
\getchunk{defun leaveScratchpad}
\getchunk{defun letPrint}
\getchunk{defun letPrint2}
\getchunk{defun letPrint3}
\getchunk{defun lfkey}
\getchunk{defun library}
\getchunk{defun line?}
\getchunk{defun lineoftoks}
\getchunk{defun listConstructorAbbreviations}
\getchunk{defun listDecideHowMuch}
\getchunk{defun listOutputter}
\getchunk{defun lnFileName}
\getchunk{defun load}
\getchunk{defun loadFunctor}
\getchunk{defun loadLib}
\getchunk{defun loadLibNoUpdate}
\getchunk{defun localdatabase}
\getchunk{defun localnrlib}
\getchunk{defun lookupInDomainVector}
\getchunk{defun loopIters2Sex}
\getchunk{defun lotsof}
\getchunk{defun ltrace}

\getchunk{defun macApplication}
\getchunk{defun macExpand}
\getchunk{defun macId}
\getchunk{defun macLambda}
```

```
\getchunk{defun macLambda,mac}
\getchunk{defun macLambdaParameterHandling}
\getchunk{defun macMacro}
\getchunk{defun macSubstituteId}
\getchunk{defun macSubstituteOuter}
\getchunk{defun macroExpanded}
\getchunk{defun macWhere}
\getchunk{defun macWhere,mac}
\getchunk{defun macOExpandBody}
\getchunk{defun macOGet}
\getchunk{defun macOGetName}
\getchunk{defun macOInfiniteExpansion}
\getchunk{defun macOMLambdaApply}
\getchunk{defun macOSubstituteOuter}
\getchunk{defun make-appendstream}
\getchunk{defun make-databases}
\getchunk{defun makeFullNamestring}
\getchunk{defun makeHistFileName}
\getchunk{defun makeInputFilename}
\getchunk{defun make-instream}
\getchunk{defun makeLeaderMsg}
\getchunk{defun makeMsgFromLine}
\getchunk{defun makeOrdinal}
\getchunk{defun make-outstream}
\getchunk{defun makePathname}
\getchunk{defun makeStream}
\getchunk{defun mapLetPrint}
\getchunk{defun mergePathnames}
\getchunk{defun messageprint}
\getchunk{defun messageprint-1}
\getchunk{defun messageprint-2}
\getchunk{defun mkDomPvar}
\getchunk{defun mkEvalable}
\getchunk{defun mkEvalableMapping}
\getchunk{defun mkEvalableRecord}
\getchunk{defun mkEvalableUnion}
\getchunk{defun mkLineList}
\getchunk{defun mkprompt}
\getchunk{defun msgCreate}
\getchunk{defun msgImPr?}
\getchunk{defun msgNoRep?}
\getchunk{defun msgOutputter}
\getchunk{defun msgText}
\getchunk{defun myWritable?}

\getchunk{defun namestring}
\getchunk{defun ncAlist}
\getchunk{defun ncBug}
\getchunk{defun ncConversationPhase}
\getchunk{defun ncConversationPhase,wrapup}
```

```
\getchunk{defun ncEltQ}
\getchunk{defun ncHardError}
\getchunk{defun ncIntLoop}
\getchunk{defun ncloopCommand}
\getchunk{defun ncloopDQlines}
\getchunk{defun ncloopIncFileName}
\getchunk{defun ncloopInclude}
\getchunk{defun ncloopInclude0}
\getchunk{defun ncloopInclude1}
\getchunk{defun ncloopParse}
\getchunk{defun ncParseFromString}
\getchunk{defun ncPutQ}
\getchunk{defun ncSoftError}
\getchunk{defun ncTag}
\getchunk{defun ncTopLevel}
\getchunk{defun newHelpSpad2Cmd}
\getchunk{defun next}
\getchunk{defun next1}
\getchunk{defun nextInterpreterFrame}
\getchunk{defun nextline}
\getchunk{defun next-lines-clear}
\getchunk{defun next-lines-show}
\getchunk{defun npAdd}
\getchunk{defun npADD}
\getchunk{defun npAmpersand}
\getchunk{defun npAmpersandFrom}
\getchunk{defun npAndOr}
\getchunk{defun npAngleBared}
\getchunk{defun npApplication}
\getchunk{defun npApplication2}
\getchunk{defun npArith}
\getchunk{defun npAssign}
\getchunk{defun npAssignment}
\getchunk{defun npAssignVariable}
\getchunk{defun npAtom1}
\getchunk{defun npAtom2}
\getchunk{defun npBacksetElse}
\getchunk{defun npBackTrack}
\getchunk{defun npBDefinition}
\getchunk{defun npBPileDefinition}
\getchunk{defun npBraced}
\getchunk{defun npBracked}
\getchunk{defun npBracketed}
\getchunk{defun npBreak}
\getchunk{defun npBy}
\getchunk{defun npCategory}
\getchunk{defun npCategoryL}
\getchunk{defun npCoerceTo}
\getchunk{defun npColon}
\getchunk{defun npColonQuery}
```

```
\getchunk{defun npComma}
\getchunk{defun npCommaBackSet}
\getchunk{defun npCompMissing}
\getchunk{defun npConditional}
\getchunk{defun npConditionalStatement}
\getchunk{defun npConstTok}
\getchunk{defun npDDInfKey}
\getchunk{defun npDecl}
\getchunk{defun npDef}
\getchunk{defun npDefaultDecl}
\getchunk{defun npDefaultItem}
\getchunk{defun npDefaultItemlist}
\getchunk{defun npDefaultValue}
\getchunk{defun npDefinition}
\getchunk{defun npDefinitionItem}
\getchunk{defun npDefinitionlist}
\getchunk{defun npDefinitionOrStatement}
\getchunk{defun npDefn}
\getchunk{defun npDefTail}
\getchunk{defun npDiscrim}
\getchunk{defun npDisjand}
\getchunk{defun npDollar}
\getchunk{defun npDotted}
\getchunk{defun npElse}
\getchunk{defun npEncAp}
\getchunk{defun npEncl}
\getchunk{defun npEnclosed}
\getchunk{defun npEqKey}
\getchunk{defun npExit}
\getchunk{defun npExpress}
\getchunk{defun npExpress1}
\getchunk{defun npExport}
\getchunk{defun npFirstTok}
\getchunk{defun npFix}
\getchunk{defun npForIn}
\getchunk{defun npFree}
\getchunk{defun npFromdom}
\getchunk{defun npFromdom1}
\getchunk{defun npGives}
\getchunk{defun npId}
\getchunk{defun npImport}
\getchunk{defun npInfGeneric}
\getchunk{defun npInfixOp}
\getchunk{defun npInfixOperator}
\getchunk{defun npInfKey}
\getchunk{defun npInline}
\getchunk{defun npInterval}
\getchunk{defun npItem}
\getchunk{defun npItem1}
\getchunk{defun npIterate}
```

```
\getchunk{defun npIterator}
\getchunk{defun npIterators}
\getchunk{defun npLambda}
\getchunk{defun npLeftAssoc}
\getchunk{defun npLet}
\getchunk{defun npLetQualified}
\getchunk{defun npList}
\getchunk{defun npListAndRecover}
\getchunk{defun npListing}
\getchunk{defun npListofFun}
\getchunk{defun npLocal}
\getchunk{defun npLocalDecl}
\getchunk{defun npLocalItem}
\getchunk{defun npLocalItemlist}
\getchunk{defun npLogical}
\getchunk{defun npLoop}
\getchunk{defun npMacro}
\getchunk{defun npMatch}
\getchunk{defun npMdef}
\getchunk{defun npMDEF}
\getchunk{defun npMDEFinition}
\getchunk{defun npMissing}
\getchunk{defun npMissingMate}
\getchunk{defun npMoveTo}
\getchunk{defun npName}
\getchunk{defun npNext}
\getchunk{defun npNull}
\getchunk{defun npParened}
\getchunk{defun npParenthesize}
\getchunk{defun npParenthesized}
\getchunk{defun npParse}
\getchunk{defun npPDefinition}
\getchunk{defun npPileBracketed}
\getchunk{defun npPileDefinitionlist}
\getchunk{defun npPileExit}
\getchunk{defun npPower}
\getchunk{defun npPP}
\getchunk{defun npPPf}
\getchunk{defun npPPff}
\getchunk{defun npPPg}
\getchunk{defun npPrefixColon}
\getchunk{defun npPretend}
\getchunk{defun npPrimary}
\getchunk{defun npPrimary1}
\getchunk{defun npPrimary2}
\getchunk{defun npProcessSynonym}
\getchunk{defun npProduct}
\getchunk{defun npPushId}
\getchunk{defun npRelation}
\getchunk{defun npRemainder}
```

```
\getchunk{defun npQualDef}
\getchunk{defun npQualified}
\getchunk{defun npQualifiedDefinition}
\getchunk{defun npQualType}
\getchunk{defun npQualTypelist}
\getchunk{defun npQuiver}
\getchunk{defun npRecoverTrap}
\getchunk{defun npRestore}
\getchunk{defun npRestrict}
\getchunk{defun npReturn}
\getchunk{defun npRightAssoc}
\getchunk{defun npRule}
\getchunk{defun npSCategory}
\getchunk{defun npSDefaultItem}
\getchunk{defun npSegment}
\getchunk{defun npSelector}
\getchunk{defun npSemiBackSet}
\getchunk{defun npSemiListing}
\getchunk{defun npSigDecl}
\getchunk{defun npSigItem}
\getchunk{defun npSigItemlist}
\getchunk{defun npSignature}
\getchunk{defun npSignatureDefinee}
\getchunk{defun npSingleRule}
\getchunk{defun npSLocalItem}
\getchunk{defun npSQualTypelist}
\getchunk{defun npStatement}
\getchunk{defun npSuch}
\getchunk{defun npSuchThat}
\getchunk{defun npSum}
\getchunk{defun npsynonym}
\getchunk{defun npSymbolVariable}
\getchunk{defun npSynthetic}
\getchunk{defun npsystem}
\getchunk{defun npState}
\getchunk{defun npTagged}
\getchunk{defun npTerm}
\getchunk{defun npTrap}
\getchunk{defun npTrapForm}
\getchunk{defun npTuple}
\getchunk{defun npType}
\getchunk{defun npTypedForm}
\getchunk{defun npTypedForm1}
\getchunk{defun npTypeStyle}
\getchunk{defun npTypified}
\getchunk{defun npTyping}
\getchunk{defun npTypeVariable}
\getchunk{defun npTypeVariablelist}
\getchunk{defun npVariable}
\getchunk{defun npVariablelist}
```

```
\getchunk{defun npVariableName}
\getchunk{defun npVoid}
\getchunk{defun npWConditional}
\getchunk{defun npWhile}
\getchunk{defun npWith}
\getchunk{defun npZeroOrMore}
\getchunk{defun NRTevalDomain}

\getchunk{defun ofCategory}
\getchunk{defun oldCompLookup}
\getchunk{defun oldHistFileName}
\getchunk{defun om-bindTCP}
\getchunk{defun om-closeConn}
\getchunk{defun om-closeDev}
\getchunk{defun om-connectTCP}
\getchunk{defun om-getApp}
\getchunk{defun om-getAtp}
\getchunk{defun om-getAttr}
\getchunk{defun om-getBind}
\getchunk{defun om-getBVar}
\getchunk{defun om-getConnInDev}
\getchunk{defun om-getConnOutDev}
\getchunk{defun om-getEndApp}
\getchunk{defun om-getEndAtp}
\getchunk{defun om-getEndAttr}
\getchunk{defun om-getEndBind}
\getchunk{defun om-getEndBVar}
\getchunk{defun om-getEndError}
\getchunk{defun om-getEndObject}
\getchunk{defun om-getError}
\getchunk{defun om-getFloat}
\getchunk{defun om-getInt}
\getchunk{defun om-getObject}
\getchunk{defun om-getString}
\getchunk{defun om-getSymbol}
\getchunk{defun om-getType}
\getchunk{defun om-getVar}
\getchunk{defun om-listCDs}
\getchunk{defun om-listSymbols}
\getchunk{defun om-makeConn}
\getchunk{defun om-openFileDev}
\getchunk{defun om-openStringDev}
\getchunk{defun om-putApp}
\getchunk{defun om-putAtp}
\getchunk{defun om-putAttr}
\getchunk{defun om-putBind}
\getchunk{defun om-putBVar}
\getchunk{defun om-putByteArray}
\getchunk{defun om-putEndApp}
\getchunk{defun om-putEndAtp}
```

```
\getchunk{defun om-putEndAttr}
\getchunk{defun om-putEndBind}
\getchunk{defun om-putEndBVar}
\getchunk{defun om-putEndError}
\getchunk{defun om-putEndObject}
\getchunk{defun om-putError}
\getchunk{defun om-putFloat}
\getchunk{defun om-putInt}
\getchunk{defun om-putObject}
\getchunk{defun om-putString}
\getchunk{defun om-putSymbol}
\getchunk{defun om-putVar}
\getchunk{defun om-Read}
\getchunk{defun om-setDevEncoding}
\getchunk{defun om-stringPtrToString}
\getchunk{defun om-stringToStringPtr}
\getchunk{defun om-supportsCD}
\getchunk{defun om-supportsSymbol}
\getchunk{defun openOutputLibrary}
\getchunk{defun openserver}
\getchunk{defun operationOpen}
\getchunk{defun optionError}
\getchunk{defun optionUserLevelError}
\getchunk{defun orderBySlotNumber}

\getchunk{defun parseAndInterpret}
\getchunk{defun parseFromString}
\getchunk{defun parseSystemCmd}
\getchunk{defun pathname}
\getchunk{defun pathnameDirectory}
\getchunk{defun pathnameName}
\getchunk{defun pathnameType}
\getchunk{defun pathnameTypeId}
\getchunk{defun patternVarsOf}
\getchunk{defun patternVarsOf1}
\getchunk{defun pcounters}
\getchunk{defun pfAbSynOp}
\getchunk{defun pfAbSynOp?}
\getchunk{defun pfAdd}
\getchunk{defun pfAnd}
\getchunk{defun pfAnd?}
\getchunk{defun pfApplication}
\getchunk{defun pfApplication?}
\getchunk{defun pfApplication2Sex}
\getchunk{defun pfAssign}
\getchunk{defun pfAssign?}
\getchunk{defun pfAttribute}
\getchunk{defun pfBrace}
\getchunk{defun pfBraceBar}
\getchunk{defun pfBracket}
```

```
\getchunk{defun pfBracketBar}
\getchunk{defun pfBreak}
\getchunk{defun pfBreak?}
\getchunk{defun pfCharPosn}
\getchunk{defun pfCheckArg}
\getchunk{defun pfCheckMacroOut}
\getchunk{defun pfCheckId}
\getchunk{defun pfCheckItOut}
\getchunk{defun pfCoerceto}
\getchunk{defun pfCoerceto?}
\getchunk{defun pfCollect}
\getchunk{defun pfCollect?}
\getchunk{defun pfCollect1?}
\getchunk{defun pfCollectArgTran}
\getchunk{defun pfCollectVariable1}
\getchunk{defun pfCollect2Sex}
\getchunk{defun pfCopyWithPos}
\getchunk{defun pfDefinition}
\getchunk{defun pfDefinition?}
\getchunk{defun pfDefinition2Sex}
\getchunk{defun pfDo}
\getchunk{defun pfDo?}
\getchunk{defun pfDocument}
\getchunk{defun pfEnSequence}
\getchunk{defun pfExit}
\getchunk{defun pfExit?}
\getchunk{defun pfExport}
\getchunk{defun pfExpression}
\getchunk{defun pfFileName}
\getchunk{defun pfFix}
\getchunk{defun pfFlattenApp}
\getchunk{defun pfFree}
\getchunk{defun pfFree?}
\getchunk{defun pfForin}
\getchunk{defun pfForin?}
\getchunk{defun pfFromDom}
\getchunk{defun pfFromdom}
\getchunk{defun pfFromdom?}
\getchunk{defun pfGlobalLinePosn}
\getchunk{defun pfHide}
\getchunk{defun pfId}
\getchunk{defun pfId?}
\getchunk{defun pfIdPos}
\getchunk{defun pfIdSymbol}
\getchunk{defun pfIf}
\getchunk{defun pfIf?}
\getchunk{defun pfIfThenOnly}
\getchunk{defun pfImport}
\getchunk{defun pfInline}
\getchunk{defun pfInfApplication}
```

```
\getchunk{defun pfIterate}
\getchunk{defun pfIterate?}
\getchunk{defun pfLam}
\getchunk{defun pfLambda}
\getchunk{defun pfLambdaTran}
\getchunk{defun pfLambda?}
\getchunk{defun pfLambda2Sex}
\getchunk{defun pfLeaf}
\getchunk{defun pfLeaf?}
\getchunk{defun pfLeafPosition}
\getchunk{defun pfLeafToken}
\getchunk{defun pfLhsRule2Sex}
\getchunk{defun pfLinePosn}
\getchunk{defun pfListOf}
\getchunk{defun pfLiteralClass}
\getchunk{defun pfLiteralString}
\getchunk{defun pfLiteral2Sex}
\getchunk{defun pfLocal}
\getchunk{defun pfLocal?}
\getchunk{defun pfLoop}
\getchunk{defun pfLoop1}
\getchunk{defun pfLoop?}
\getchunk{defun pfLp}
\getchunk{defun pfMacro}
\getchunk{defun pfMacro?}
\getchunk{defun pfMapParts}
\getchunk{defun pfMLambda}
\getchunk{defun pfMLambda?}
\getchunk{defun pfname}
\getchunk{defun pfNoPosition}
\getchunk{defun pfNoPosition?}
\getchunk{defun pfNot?}
\getchunk{defun pfNothing}
\getchunk{defun pfNothing?}
\getchunk{defun pfNovalue}
\getchunk{defun pfNovalue?}
\getchunk{defun pfOp2Sex}
\getchunk{defun pfOr}
\getchunk{defun pfOr?}
\getchunk{defun pfParen}
\getchunk{defun pfPretend}
\getchunk{defun pfPretend?}
\getchunk{defun pfPushBody}
\getchunk{defun pfPushMacroBody}
\getchunk{defun pfQualType}
\getchunk{defun pfRestrict}
\getchunk{defun pfRestrict?}
\getchunk{defun pfRetractTo}
\getchunk{defun pfReturn}
\getchunk{defun pfReturn?}
```

```
\getchunk{defun pfReturnNoName}
\getchunk{defun pfReturnTyped}
\getchunk{defun pfRhsRule2Sex}
\getchunk{defun pfRule}
\getchunk{defun pfRule?}
\getchunk{defun pfRule2Sex}
\getchunk{defun pfSequence}
\getchunk{defun pfSequence?}
\getchunk{defun pfSequenceToList}
\getchunk{defun pfSequence2Sex}
\getchunk{defun pfSequence2Sex0}
\getchunk{defun pfSexpr}
\getchunk{defun pfSexpr,strip}
\getchunk{defun pfSourcePosition}
\getchunk{defun pfSourceStok}
\getchunk{defun pfSpread}
\getchunk{defun pfSuch}
\getchunk{defun pfSuchthat}
\getchunk{defun pfSuchthat?}
\getchunk{defun pfSuchThat2Sex}
\getchunk{defun pfSymb}
\getchunk{defun pfSymbol}
\getchunk{defun pfSymbol?}
\getchunk{defun pfSymbolSymbol}
\getchunk{defun pfTagged}
\getchunk{defun pfTagged?}
\getchunk{defun pfTaggedToTyped}
\getchunk{defun pfTaggedToTyped1}
\getchunk{defun pfTransformArg}
\getchunk{defun pfTuple}
\getchunk{defun pfTupleListOf}
\getchunk{defun pfTweakIf}
\getchunk{defun pfTyped}
\getchunk{defun pfTyped?}
\getchunk{defun pfTyping}
\getchunk{defun pfTuple?}
\getchunk{defun pfUnSequence}
\getchunk{defun pfWDec}
\getchunk{defun pfWDeclare}
\getchunk{defun pfWhere}
\getchunk{defun pfWhere?}
\getchunk{defun pfWhile}
\getchunk{defun pfWhile?}
\getchunk{defun pfWith}
\getchunk{defun pfWrong}
\getchunk{defun pfWrong?}
\getchunk{defun pf0ApplicationArgs}
\getchunk{defun pf0DefinitionLhsItems}
\getchunk{defun pf0FlattenSyntacticTuple}
\getchunk{defun pf0ForinLhs}
```

```
\getchunk{defun pf0FreeItems}
\getchunk{defun pf0LambdaArgs}
\getchunk{defun pf0LocalItems}
\getchunk{defun pf0LoopIterators}
\getchunk{defun pf0MLambdaArgs}
\getchunk{defun pf0SequenceArgs}
\getchunk{defun pf0TupleParts}
\getchunk{defun pf0WhereContext}
\getchunk{defun pf2Sex}
\getchunk{defun pf2Sex1}
\getchunk{defun phMacro}
\getchunk{defun phParse}
\getchunk{defun phInterpret}
\getchunk{defun phIntReportMsgs}
\getchunk{defun pileCforest}
\getchunk{defun pileColumn}
\getchunk{defun pileCtree}
\getchunk{defun pileForest}
\getchunk{defun pileForest1}
\getchunk{defun pileForests}
\getchunk{defun pilePlusComment}
\getchunk{defun pilePlusComments}
\getchunk{defun pileTree}
\getchunk{defun poFileName}
\getchunk{defun poGlobalLinePosn}
\getchunk{defun poLinePosn}
\getchunk{defun poPosImmediate?}
\getchunk{defun porigin}
\getchunk{defun posend}
\getchunk{defun posPointers}
\getchunk{defun ppos}
\getchunk{defun pquit}
\getchunk{defun pquitSpad2Cmd}
\getchunk{defun previousInterpreterFrame}
\getchunk{defun printLabelledList}
\getchunk{defun printStatisticsSummary}
\getchunk{defun printStorage}
\getchunk{defun printSynonyms}
\getchunk{defun printTypeAndTime}
\getchunk{defun printTypeAndTimeNormal}
\getchunk{defun printTypeAndTimeSaturn}
\getchunk{defun probeName}
\getchunk{defun processChPosesForOneLine}
\getchunk{defun processInteractive}
\getchunk{defun processInteractive1}
\getchunk{defun processKeyedError}
\getchunk{defun processMsgList}
\getchunk{defun protectedEVAL}
\getchunk{defun processSynonymLine}
\getchunk{defun processSynonymLine,removeKeyFromLine}
```

```
\getchunk{defun processSynonyms}
\getchunk{defun prTraceNames}
\getchunk{defun prTraceNames,fn}
\getchunk{defun pspacers}
\getchunk{defun ptimers}
\getchunk{defun put}
\getchunk{defun putFTText}
\getchunk{defun punctuation?}
\getchunk{defun putDatabaseStuff}
\getchunk{defun putHist}
\getchunk{defun pvarPredTran}

\getchunk{defun queryClients}
\getchunk{defun queueUpErrors}
\getchunk{defun quit}
\getchunk{defun quitSpad2Cmd}

\getchunk{defun rassocSub}
\getchunk{defun rdefinstream}
\getchunk{defun rdefoutstream}
\getchunk{defun read}
\getchunk{defun /read}
\getchunk{defun readHiFi}
\getchunk{defun readSpadProfileIfThere}
\getchunk{defun readSpad2Cmd}
\getchunk{defun recordAndPrint}
\getchunk{defun recordFrame}
\getchunk{defun recordNewValue}
\getchunk{defun recordNewValue0}
\getchunk{defun recordOldValue}
\getchunk{defun recordOldValue0}
\getchunk{defun redundant}
\getchunk{defun remFile}
\getchunk{defun removeOption}
\getchunk{defun remover}
\getchunk{defun removeTracedMapSigs}
\getchunk{defun removeUndoLines}
\getchunk{defun replaceFile}
\getchunk{defun replaceSharps}
\getchunk{defun reportOperations}
\getchunk{defun reportOpsFromLisplib}
\getchunk{defun reportOpsFromLisplib0}
\getchunk{defun reportOpsFromLisplib1}
\getchunk{defun reportOpsFromUnitDirectly}
\getchunk{defun reportOpsFromUnitDirectly0}
\getchunk{defun reportOpsFromUnitDirectly1}
\getchunk{defun reportSpadTrace}
\getchunk{defun reportUndo}
\getchunk{defun reportWhatOptions}
\getchunk{defun reroot}
```

```
\getchunk{defun resetCounters}
\getchunk{defun resethashtables}
\getchunk{defun resetInCoreHist}
\getchunk{defun resetSpacers}
\getchunk{defun resetTimers}
\getchunk{defun resetWorkspaceVariables}
\getchunk{defun restart}
\getchunk{defun restart0}
\getchunk{defun restoreHistory}
\getchunk{defun retract}
\getchunk{defun rread}
\getchunk{defun ruleLhsTran}
\getchunk{defun rulePredicateTran}
\getchunk{defun runspad}
\getchunk{defun rwrite}

\getchunk{defun safeWritify}
\getchunk{defun sameMsg?}
\getchunk{defun satisfiesRegularExpressions}
\getchunk{defun saveHistory}
\getchunk{defun saveMapSig}
\getchunk{defun savesystem}
\getchunk{defun saveDependentsHashTable}
\getchunk{defun saveUsersHashTable}
\getchunk{defun sayAllCacheCounts}
\getchunk{defun sayBrightly1}
\getchunk{defun sayCacheCount}
\getchunk{defun sayExample}
\getchunk{defun sayKeyedMsg}
\getchunk{defun sayKeyedMsgLocal}
\getchunk{defun sayMSG}
\getchunk{defun sayMSG2File}
\getchunk{defun sayShowWarning}
\getchunk{defun scanCheckRadix}
\getchunk{defun scanComment}
\getchunk{defun scanDictCons}
\getchunk{defun scanError}
\getchunk{defun scanEsc}
\getchunk{defun scanEscape}
\getchunk{defun scanExponent}
\getchunk{defun scanIgnoreLine}
\getchunk{defun scanInsert}
\getchunk{defun scanKeyTr}
\getchunk{defun scanNegComment}
\getchunk{defun scanNumber}
\getchunk{defun ScanOrPairVec}
\getchunk{defun ScanOrPairVec,ScanOrInner}
\getchunk{defun scanPossFloat}
\getchunk{defun scanPunct}
\getchunk{defun scanPunCons}
```

```
\getchunk{defun scanS}
\getchunk{defun scanSpace}
\getchunk{defun scanString}
\getchunk{defun scanKeyTableCons}
\getchunk{defun scanToken}
\getchunk{defun scanTransform}
\getchunk{defun scanW}
\getchunk{defun scanWord}
\getchunk{defun search}
\getchunk{defun searchCurrentEnv}
\getchunk{defun searchTailEnv}
\getchunk{defun segmentKeyedMsg}
\getchunk{defun selectOption}
\getchunk{defun selectOptionLC}
\getchunk{defun separatePiles}
\getchunk{defun serverReadLine}
\getchunk{defun set}
\getchunk{defun set1}
\getchunk{defun setdatabase}
\getchunk{defun setExpose}
\getchunk{defun setExposeAdd}
\getchunk{defun setExposeAddConstr}
\getchunk{defun setExposeAddGroup}
\getchunk{defun setExposeDrop}
\getchunk{defun setExposeDropConstr}
\getchunk{defun setExposeDropGroup}
\getchunk{defun setFortDir}
\getchunk{defun setFortPers}
\getchunk{defun setFortTmpDir}
\getchunk{defun setFunctionsCache}
\getchunk{defun setHistoryCore}
\getchunk{defun setInputLibrary}
\getchunk{defun setIOindex}
\getchunk{defun setLinkerArgs}
\getchunk{defun setMsgCatlessAttr}
\getchunk{defun setMsgForcedAttr}
\getchunk{defun setMsgForcedAttrList}
\getchunk{defun setMsgUnforcedAttr}
\getchunk{defun setMsgUnforcedAttrList}
\getchunk{defun setNagHost}
\getchunk{defun setOutputAlgebra}
\getchunk{defun setOutputCharacters}
\getchunk{defun setOutputFormula}
\getchunk{defun setOutputFortran}
\getchunk{defun setOutputLibrary}
\getchunk{defun setOutputHtml}
\getchunk{defun setOutputMathml}
\getchunk{defun setOutputOpenMath}
\getchunk{defun setOutputTex}
\getchunk{defun setStreamsCalculate}
```

```
\getchunk{defun shortenForPrinting}
\getchunk{defun show}
\getchunk{defun showdatabase}
\getchunk{defun showInOut}
\getchunk{defun showInput}
\getchunk{defun showSpad2Cmd}
\getchunk{defun shut}
\getchunk{defun size}
\getchunk{defun SkipEnd?}
\getchunk{defun SkipPart?}
\getchunk{defun Skipping?}
\getchunk{defun spad}
\getchunk{defun spadClosure?}
\getchunk{defun spad-error-loc}
\getchunk{defun SpadInterpretStream}
\getchunk{defun spad-long-error}
\getchunk{defun spadReply}
\getchunk{defun spadReply,printName}
\getchunk{defun spadrread}
\getchunk{defun spadrwrite}
\getchunk{defun spadrwrite0}
\getchunk{defun spad-save}
\getchunk{defun spad-short-error}
\getchunk{defun spadStartUpMsgs}
\getchunk{defun spad-syntax-error}
\getchunk{defun spadTrace}
\getchunk{defun spadTraceAlias}
\getchunk{defun spadTrace,g}
\getchunk{defun spadTrace,isTraceable}
\getchunk{defun spadUntrace}
\getchunk{defun spad2BootCoerce}
\getchunk{defun specialChar}
\getchunk{defun spleI}
\getchunk{defun spleI1}
\getchunk{defun splitIntoOptionBlocks}
\getchunk{defun squeeze}
\getchunk{defun stackTraceOptionError}
\getchunk{defun startsComment?}
\getchunk{defun startsNegComment?}
\getchunk{defun statisticsInitialization}
\getchunk{defun streamChop}
\getchunk{defun stringMatches?}
\getchunk{defun StringToDir}
\getchunk{defun strpos}
\getchunk{defun strposl}
\getchunk{defun stupidIsSpadFunction}
\getchunk{defun subMatch}
\getchunk{defun substringMatch}
\getchunk{defun subTypes}
\getchunk{defun summary}
```

```
\getchunk{defun syGeneralErrorHere}
\getchunk{defun syIgnoredFromTo}
\getchunk{defun synonym}
\getchunk{defun synonymsForUserLevel}
\getchunk{defun synonymSpad2Cmd}
\getchunk{defun sySpecificErrorAtToken}
\getchunk{defun sySpecificErrorHere}
\getchunk{defun systemCommand}

\getchunk{defun ?t}
\getchunk{defun tabbing}
\getchunk{defun terminateSystemCommand}
\getchunk{defun tersyscommand}
\getchunk{defun thisPosIsEqual}
\getchunk{defun thisPosIsLess}
\getchunk{defun throwEvalTypeMsg}
\getchunk{defun toFile?}
\getchunk{defun tokConstruct}
\getchunk{defun token-stack-show}
\getchunk{defun tokPosn}
\getchunk{defun tokTran}
\getchunk{defun tokType}
\getchunk{defun toScreen?}
\getchunk{defun trace}
\getchunk{defun trace1}
\getchunk{defun traceDomainConstructor}
\getchunk{defun traceDomainLocalOps}
\getchunk{defun tracelet}
\getchunk{defun traceOptionError}
\getchunk{defun /tracereply}
\getchunk{defun traceReply}
\getchunk{defun traceSpad2Cmd}
\getchunk{defun translateTrueFalse2YesNo}
\getchunk{defun translateYesNo2TrueFalse}
\getchunk{defun transOnlyOption}
\getchunk{defun transTraceItem}

\getchunk{defun unAbbreviateKeyword}
\getchunk{defun undo}
\getchunk{defun undoChanges}
\getchunk{defun undoCount}
\getchunk{defun undoFromFile}
\getchunk{defun undoInCore}
\getchunk{defun undoLocalModemapHack}
\getchunk{defun undoSingleStep}
\getchunk{defun undoSteps}
\getchunk{defun unescapeStringsInForm}
\getchunk{defun unifyStruct}
\getchunk{defun unifyStructVar}
\getchunk{defun unsqueeze}
```

```
\getchunk{defun untrace}
\getchunk{defun untraceDomainConstructor}
\getchunk{defun untraceDomainConstructor,keepTraced?}
\getchunk{defun untraceDomainLocalOps}
\getchunk{defun untraceMapSubNames}
\getchunk{defun unwritable?}
\getchunk{defun updateCurrentInterpreterFrame}
\getchunk{defun updateDatabase}
\getchunk{defun updateFromCurrentInterpreterFrame}
\getchunk{defun updateHist}
\getchunk{defun updateInCoreHist}
\getchunk{defun updateSourceFiles}
\getchunk{defun userLevelErrorMessage}

\getchunk{defun validateOutputDirectory}
\getchunk{defun vec2list}
\getchunk{defun voidValue}

\getchunk{defun what}
\getchunk{defun whatCommands}
\getchunk{defun whatConstructors}
\getchunk{defun whatSpad2Cmd}
\getchunk{defun whatSpad2Cmd,fixpat}
\getchunk{defun whichCat}
\getchunk{defun with}
\getchunk{defun workfiles}
\getchunk{defun workfilesSpad2Cmd}
\getchunk{defun wrap}
\getchunk{defun write-browsedb}
\getchunk{defun write-categorydb}
\getchunk{defun write-compress}
\getchunk{defun writeHiFi}
\getchunk{defun writeHistModesAndValues}
\getchunk{defun writeInputLines}
\getchunk{defun write-interpdb}
\getchunk{defun write-operationdb}
\getchunk{defun write-warmdata}
\getchunk{defun writify}
\getchunk{defun writifyComplain}
\getchunk{defun writify,writifyInner}

\getchunk{defun xlCannotRead}
\getchunk{defun xlCmdBug}
\getchunk{defun xlConActive}
\getchunk{defun xlConsole}
\getchunk{defun xlConStill}
\getchunk{defun xlFileCycle}
\getchunk{defun xlIfBug}
\getchunk{defun xlIfSyntax}
\getchunk{defun xlMsg}
```

```
\getchunk{defun xlNoSuchFile}
\getchunk{defun xlOK}
\getchunk{defun xlOK1}
\getchunk{defun xlPrematureEOF}
\getchunk{defun xlPrematureFin}
\getchunk{defun xlSay}
\getchunk{defun xlSkip}
\getchunk{defun xlSkippingFin}

\getchunk{defun yesanswer}

\getchunk{defun zsystemdevelopment}
\getchunk{defun zsystemdevelopment1}
\getchunk{defun zsystemDevelopmentSpad2Cmd}

\getchunk{postvars}
```

———-

Chapter 74

The Global Variables

74.1 Star Global Variables

NAME	SET	USE
eof*	ncTopLevel	
features*		restart
package*		restart
standard-input*		ncIntLoop
standard-output*		ncIntLoop
top-level-hook*	set-restart-hook	

74.1.1 *eof*

The `*eof*` variable is set to NIL in ncTopLevel.

74.1.2 *features*

The `*features*` variable from common lisp is tested for the presence of the `:unix` keyword. Apparently this controls the use of Saturn, a previous Axiom frontend. The Saturn frontend was never released as open source and so this test and the associated variables are probably not used.

74.1.3 *package*

The `*package*` variable, from common lisp, is set in restart to the BOOT package where the intepreter lives.

74.1.4 *standard-input*

The `*standard-input*` common lisp variable is used to set the curinstream variable in ncIntLoop.

This variable is an argument to serverReadLine in the intloopReadConsole function.

74.1.5 *standard-output*

The `*standard-output*` common lisp variable is used to set the curoutstream variable in ncIntLoop.

74.1.6 *top-level-hook*

The `*top-level-hook*` common lisp variable contains the name of a function to invoke when an image is started. In our case it is called restart. This is the entry point to the Axiom interpreter.

74.2 Dollar Global Variables

NAME	SET	USE
$boot	ncTopLevel	
coerceFailure		runspad
curinstream	ncIntLoop	
curoutstream	ncIntLoop	
$currentLine	restart	removeUndoLines
$dalymode		intloopReadConsole
$displayStartMsgs		restart
$e	ncTopLevel	
$erMsgToss	SpadInterpretStream	
$fn	SpadInterpretStream	
$frameRecord	initvars	
	clearFrame	
	undoSteps	undoSteps
	recordFrame	recordFrame
$HiFiAccess	initHist	historySpad2Cmd
	historySpad2Cmd	
		setHistoryCore
$HistList	initHist	
$HistListAct	initHist	
$HistListLen	initHistList	
$HistRecord	initHistList	
$historyDirectory		makeHistFileName
		makeHistFileName
$historyFileType	initvars	histInputFileName
$InteractiveFrame	restart	ncTopLevel
	undo	recordFrame
	undoSteps	undoSteps
		reportUndo
$internalHistoryTable	initvars	
$interpreterFrameName	initializeInterpreterFrameRing	
$interpreterFrameRing	initializeInterpreterFrameRing	
$intRestart		intloop
$intTopLevel	intloop	
$IOindex	restart	historySpad2Cmd
	removeUndoLines	undoCount
$genValue	bookvol5	i-toplev
		i-analy
		i-syscmd
		i-spec1
		i-spec2
		i-map
$lastPos	SpadInterpretStream	
$libQuiet	SpadInterpretStream	
$msgDatabaseName	reroot *	
$ncMsgList	SpadInterpretStream	
$newcompErrorCount	SpadInterpretStream	
$newspad	ncTopLevel	
$nopos		SpadInterpretStream
$okToExecuteMachineCode	SpadInterpretStream	
$oldHistoryFileName	initvars	oldHistFileName
$options		history
	historySpad2Cmd	historySpad2Cmd
		undo

74.2.1 $boot

The `$boot` variable is set to NIL in ncTopLevel.

74.2.2 coerceFailure

The coerceFailure symbol is a catch tag used in runspad to catch an exit from ncTopLevel.

74.2.3 $currentLine

The `$currentLine` line is set to NIL in restart. It is used in removeUndoLines in the undo mechanism.

74.2.4 $displayStartMsgs

The `$displayStartMsgs` variable is used in restart but is not set so this is likely a bug.

74.2.5 $e

The `$e` variable is set to the value of `$InteractiveFrame` which is set in restart to the value of the call to the makeInitialModemapFrame function. This function simply returns a copy of the variable `$InitialModemapFrame`.

Thus `$e` is a copy of the variable `$InitialModemapFrame`.

This variable is used in the undo mechanism.

74.2.6 $erMsgToss

The `$erMsgToss` variable is set to NIL in SpadInterpretStream.

74.2.7 $fn

The `$fn` variable is set in SpadInterpretStream. It is set to the second argument which is a list. It appears that this list has the same structure as an argument to the LispVM rdefiostream function.

74.2.8 $frameRecord

`$frameRecord = [delta1, delta2,... ]` where delta(i) contains changes in the "backwards" direction. Each delta(i) has the form ((var . proplist)...) where proplist denotes an ordinary proplist. For example, an entry of the form ((x (value) (mode (Integer)))...) indicates that to undo 1 step, x's value is cleared and its mode should be set to (Integer).

A delta(i) of the form (systemCommand . delta) is a special delta indicating changes due to system commands executed between the last command and the current command. By recording these deltas separately, it is possible to undo to either BEFORE or AFTER the command. These special delta(i)s are given ONLY when a a system command is given which alters the environment.

Note: recordFrame('system) is called before a command is executed, and recordFrame('normal) is called after (see processInteractive1). If no changes are found for former, no special entry is given.

This is part of the undo mechanism.

74.2.9 $HiFiAccess

The `$HiFiAccess` is set by initHist to T. It is a flag used by the history mechanism to record whether the history function is currently on. It can be reset by using the axiom command

```
)history off
```

It appears that the name means "History File Access".

The `$HiFiAccess` variable is used by historySpad2Cmd to check whether history is turned on. T means it is, NIL means it is not.

74.2.10 $HistList

Thie `$HistList` variable is set by initHistList to an initial value of NIL elements. The last element of the list is smashed to point to the first element to make the list circular. This is a circular list of length `$HistListLen`.

74.2.11 $HistListAct

The `$HistListAct` variable is set by initHistList to 0. This variable holds the actual number of elements in the history list. This is the number of "undoable" steps.

74.2.12 $HistListLen

The `$HistListLen` variable is set by initHistList to 20. This is the length of a circular list maintained in the variable `$HistList`.

74.2.13 $HistRecord

The `$HistRecord` variable is set by initHistList to NIL. `$HistRecord` collects the input line, all variable bindings and the output of a step, before it is written to the file named by the function histFileName.

74.2.14 $historyFileType

The `$historyFileType` is set at load time by a call to initvars to a value of "axh". It appears that this is intended to be used as a filetype extension. It is part of the history mechanism. It is used in makeHistFileName as part of the history file name.

74.2.15 $internalHistoryTable

The `$internalHistoryTable` variable is set at load time by a call to initvars to a value of NIL. It is part of the history mechanism.

74.2.16 $interpreterFrameName

The `$interpreterFrameName` variable, set in initializeInterpreterFrameRing to the constant initial to indicate that this is the initial (default) frame.

Frames are structures that capture all of the variables defined in a session. There can be multiple frames and the user can freely switch between them. Frames are kept in a ring data structure so you can move around the ring.

74.2.17 $interpreterFrameRing

The `$interpreterFrameRing` is set to a pair whose car is set to the result of emptyInterpreter-Frame

74.2.18 $InteractiveFrame

The `$InteractiveFrame` is set in restart to the value of the call to the makeInitialModemapFrame function. This function simply returns a copy of the variable `$InitialModemapFrame`

74.2.19 $intRestart

The `$intRestart` variable is used in intloop but has no value. This is probably a bug. While the variable's value is unchanged the system will continually reenter the SpadInterpretStream function.

74.2.20 $intTopLevel

The `$intTopLevel` is a catch tag. Throwing to this tags which is caught in the intloop will restart the SpadInterpretStream function.

74.2.21 $IOindex

The `$IOindex` index variable is set to 1 in restart. This variable is used in the historySpad2Cmd function in the history mechanism. It is set in the removeUndoLines function in the undo mechanism.

This is used in the undo mechanism in function undoCount to compute the number of undos. You can't undo more actions then have already happened.

74.2.22 $lastPos

The `$lastPos` variable is set in SpadInterpretStream to the value of the `$nopos` variable. Since `$nopos` appears to have no value this is likely a bug.

74.2.23 $libQuiet

The `$libQuiet` variable is set to the third argument of the SpadInterpretStream function. This is passed from intloop with the value of T. This variable appears to be intended to control the printing of library loading messages which would need to be suppressed if input was coming from a file.

74.2.24 $msgDatabaseName

The `$msgDatabaseName` is set to NIL in reroot.

74.2.25 $ncMsgList

The `$ncMsgList` is set to NIL in SpadInterpretStream.

74.2.26 $newcompErrorCount

The `$newcompErrorCount` is set to 0 in SpadInterpretStream.

74.2.27 $newspad

The `$newspad` is set to T in ncTopLevel.

74.2.28 $nopos

The `$nopos` variable is used in SpadInterpretStream but does not appear to have a value and is likely a bug.

74.2.29 $oldHistoryFileName

The `$oldHistoryFileName` is set at load time by a call to initvars to a value of "last". It is part of the history mechanism. It is used in the function oldHistFileName and restoreHistory.

74.2.30 $okToExecuteMachineCode

The `$okToExecuteMachineCode` is set to T in SpadInterpretStream.

74.2.31 $options

The `$options` variable is tested by the history function. If it is NIL then output the message

```
You have not used the correct syntax for the history command.
Issue )help history for more information.
```

The `$options` variable is tested in the historySpad2Cmd function. It appears to record the options that were given to a spad command on the input line. The function selectOptionLC appears to take a list off options to scan.

This variable is not yet set and is probably a bug.

74.2.32 $previousBindings

The `$previousBindings` is a copy of the `CAAR $InteractiveFrame`. This is used to compute the delta(i)s stored in `$frameRecord`. This is part of the undo mechanism.

74.2.33 $PrintCompilerMessageIfTrue

The `$PrintCompilerMessageIfTrue` variable is set to NIL in spad.

74.2.34 $reportUndo

The `$reportUndo` variable is used in diffAlist. It was not normally bound but has been set to T in initvars. If the variable is set to T then we call reportUndo.

It is part of the undo mechanism.

74.2.35 $spad

The `$spad` variable is set to T in ncTopLevel.

74.2.36 $SpadServer

If an open server is not requested then this variable to T. It has no value before this time (and is thus a bug).

74.2.37 $SpadServerName

The `$SpadServerName` is passed to the openServer function, if the function exists.

74.2.38 $systemCommandFunction

The `$systemCommandFunction` is set in SpadInterpretStream to point to the function InterpExecuteSpadSystemCommand.

74.2.39 top_level

The top_level symbol is a catch tag used in runspad to catch an exit from ncTopLevel.

74.2.40 $quitTag

The `$quitTag` is used as a variable in a catch block. It appears that it can be thrown somewhere below ncTopLevel.

74.2.41 $useInternalHistoryTable

The `$useInternalHistoryTable` variable is set at load time by a call to initvars to a value of NIL. It is part of the history mechanism.

74.2.42 $undoFlag

The `$undoFlag` is used in recordFrame to decide whether to do undo recording. It is initially set to T in initvars. This is part of the undo mechanism.

Bibliography

[1] Daly, Timothy, "The Axiom Literate Documentation"
http://axiom.axiom-developer.org/axiom-website/documentation.html

[2] Daly, Timothy, "The Axiom Wiki Website"
http://axiom.axiom-developer.org

[3] Stephane Dalmas and Olivier Arsac "OpenMath" Project SAFIR, INRIA Sophia Antipolis

[4] OpenMath Technical Overview `www.openmath.org/overview/technical.html`

Chapter 75

Index

Index

allOperations, 991
    usedby allOperations, 1027
    usedby localnrlib, 1007
    defvar, 991
allconstructors, 991
    usedby allConstructors, 1027
    usedby browseOpen, 996
    usedby interpOpen, 995
    usedby localnrlib, 1007
    usedby make-databases, 1010
    usedby resethashtables, 992
    defvar, 991
ancestors-hash
    usedby write-interpdb, 1022
attributes, 1016
    usedby write-compress, 1016
    defvar, 1016
browse-stream, 990
    usedby browseOpen, 996
    usedby getdatabase, 1001
    usedby resethashtables, 991
    defvar, 990
browse-stream-stamp, 990
    usedby browseOpen, 996
    defvar, 990
build-version
    usedby axiomVersion, 472
    usedby spadStartUpMsgs, 19
category-stream, 990
    usedby categoryOpen, 997
    usedby getdatabase, 1001
    usedby resethashtables, 991
    defvar, 990
category-stream-stamp, 991
    usedby categoryOpen, 997
    usedby resethashtables, 991
    defvar, 991
compress-stream, 989
    usedby compressOpen, 1015
    usedby write-compress, 1016
    defvar, 989
compress-stream-stamp, 989
    usedby compressOpen, 1015
    usedby resethashtables, 992
    defvar, 989
compressVectorLength, 988
    usedby compressOpen, 1015
    usedby write-compress, 1016
    defvar, 988
compressvector, 988
    usedby compressOpen, 1015
    usedby make-databases, 1010
    usedby resethashtables, 992
    usedby squeeze, 1017
    usedby unsqueeze, 1018
    defvar, 988
defaultdomain-list, 987
    usedby getdatabase, 1001
    defvar, 987
eof, 24
    usedby ncTopLevel, 25
    usedby serverReadLine, 45
    defvar, 24
hasCategory-hash, 987
    usedby categoryOpen, 997
    usedby getdatabase, 1001
    usedby write-categorydb, 1025
    defvar, 987
hascategory-hash
    usedby getdatabase, 1001
    usedby resethashtables, 992
index-filename
    usedby localdatabase, 1005
interp-stream, 989

usedby getdatabase, 1001
usedby interpOpen, 995
usedby resethashtables, 991
defvar, 989
interp-stream-stamp, 989
usedby interpOpen, 995
usedby resethashtables, 992
defvar, 989
miss, 988
usedby getdatabase, 1001
defvar, 988
monitor-domains, 1147
usedby monitor-readinterp, 1160
usedby monitor-report, 1161
usedby monitor-spadfile, 1162
defvar, 1147
monitor-nrlibs, 1147
usedby monitor-dirname, 1158
defvar, 1147
monitor-table, 1148
usedby monitor-add, 1150
usedby monitor-apropos, 1163
usedby monitor-checkpoint, 1155
usedby monitor-decr, 1152
usedby monitor-delete, 1150
usedby monitor-disable, 1151
usedby monitor-enable, 1150
usedby monitor-end, 1149
usedby monitor-incr, 1152
usedby monitor-info, 1153
usedby monitor-inittable, 1148
usedby monitor-nrlib, 1159
usedby monitor-percent, 1163
usedby monitor-reset, 1151
usedby monitor-results, 1149
usedby monitor-untested, 1153
defvar, 1148
monitor-table)
usedby monitor-tested, 1154
msghash, 325
usedby cacheKeyedMsg, 327
usedby fetchKeyedMsg, 326
defvar, 325
operation-hash, 987
usedby addoperations, 999
usedby allOperations, 1028
usedby getdatabase, 1001
usedby make-databases, 1010
usedby operationOpen, 998
usedby resethashtables, 992
usedby write-operationdb, 1026
defvar, 987
operation-stream, 989
usedby getdatabase, 1001
usedby operationOpen, 998
usedby resethashtables, 991
defvar, 989
operation-stream-stamp, 990
usedby operationOpen, 998
usedby resethashtables, 992
defvar, 990
print-package
usedby monitor-checkpoint, 1155
print-pretty
usedby write-browsedb, 1024
usedby write-categorydb, 1025
usedby write-interpdb, 1022
sourcefiles
usedby make-databases, 1010
usedby resethashtables, 991
usedby write-browsedb, 1024
standard-output
usedby init-boot/spad-reader, 958
whitespace, 24
usedby assertCond, 96
defvar, 24
yearweek
usedby axiomVersion, 472
usedby spadStartUpMsgs, 19
/D,1
calledby compileBoot, 897
calledby zsystemdevelopment1, 944
/breakcondition
usedby break, 896
/comp
calledby zsystemdevelopment1, 944
/countlist
usedby pcounters, 850
usedby resetCounters, 848
usedby resetWorkspaceVariables, 646
/editfile, 511
usedby /read, 640

usedby editSpad2Cmd, 541
usedby readSpad2Cmd, 639
usedby readSpadProfileIfThere, 951
usedby resetWorkspaceVariables, 646
defvar, 511
/pretty
usedby resetWorkspaceVariables, 646
/read, 640
calledby readSpad2Cmd, 638
calls /rf[9], 640
calls /rq[9], 640
uses /editfile, 640
defun, 640
/rf[9]
called by /read, 640
/rq[9]
called by /read, 640
/sourcefiles
usedby resetWorkspaceVariables, 646
/spacelist
usedby pspacers, 849
usedby resetSpacers, 848
usedby resetWorkspaceVariables, 646
/timerlist
usedby ptimers, 849
usedby resetTimers, 848
usedby resetWorkspaceVariables, 646
/trace,0
calledby trace1, 838
/tracenames
usedby /tracereply, 884
usedby ?t, 893
usedby getBpiNameIfTracedMap, 880
usedby getMapSubNames, 859
usedby prTraceNames, 888
usedby spadReply, 885
usedby spadTrace, 867
usedby spadUntrace, 886
usedby traceReply, 890
usedby untraceDomainConstructor, 873
usedby untrace, 852
/tracereply, 884
calls devaluate, 884
calls exit, 884
calls isDomainOrPackage, 884
calls qcar, 884
calls seq, 884
uses /tracenames, 884
defun, 884
/untrace,0
calledby untraceDomainConstructor,keepTraced?, 872
calledby untrace, 852
/untrace,2
calledby untraceMapSubNames, 863
/version
usedby zsystemdevelopment1, 944
/wsname
usedby zsystemdevelopment1, 944
?t, 892
calledby trace1, 838
calls bright, 893
calls devaluate, 893
calls get, 892
calls isDomainOrPackage, 893
calls isDomain, 893
calls isgenvar, 892
calls qcar, 893
calls qcdr, 893
calls rassocSub, 893
calls reportSpadTrace, 893
calls sayBrightly, 893
calls sayMSG, 893
calls take, 893
uses /tracenames, 893
uses $InteractiveFrame, 893
uses $mapSubNameAlist, 893
defun, 892
$FINDFILE
calledby $editSpad2Cmd, 540
$erase
calledby $addNewInterpreterFrame, 557
calledby $closeInterpreterFrame, 558
$fcopy
calledby $restoreHistory, 593
$filep
calledby $setOutputAlgebra, 754
calledby $setOutputFormula, 782
calledby $setOutputFortran, 761
calledby $setOutputHtml, 772
calledby $setOutputMathml, 768
calledby $setOutputOpenMath, 777

calledby $setOutputTex, 788
$nagMessages, 749
defvar, 749
$replace
calledby $initHist, 577
$reportBottomUpFlag, 738
defvar, 738
$systemCommandFunction
calledby $intloopProcess, 64
calledby $ncloopCommand, 474
$BreakMode, 653
usedby letPrint2, 877
usedby letPrint3, 878
defvar, 653
$CallInterp
usedby serverReadLine, 45
$CatOfCatDatabase
usedby clearCmdCompletely, 498
$CategoryFrame
local def loadLibNoUpdate, 1031
local def loadLib, 1030
usedby getProplist, 954
usedby reportOpsFromUnitDirectly, 814
usedby systemCommand, 444
$CloseClient
usedby close, 507
$Coerce
local ref unifyStructVar, 427
usedby processInteractive, 50
$CommandSynonymAlist, 474
usedby npProcessSynonym, 469
usedby printSynonyms, 470
usedby processSynonyms, 34
usedby resetWorkspaceVariables, 646
usedby synonymSpad2Cmd, 824
defvar, 474
$ComplexInteger
local ref hasCateSpecialNew, 433
$ConstructorCache
usedby clearCmdSortedCaches, 497
usedby localdatabase, 1005
usedby traceDomainConstructor, 871
$CreateFrameAnswer
usedby serverReadLine, 45
$CreateFrame
usedby serverReadLine, 45
$DomOfCatDatabase
usedby clearCmdCompletely, 498
$EchoLines
usedby intloopEchoParse, 69
$EmptyEnvironment
usedby describeSpad2Cmd, 525
usedby displaySpad2Cmd, 532
$EmptyMode
local ref evaluateType1, 908
local ref evaluateType, 906
local ref hasCate, 431
local ref isPartialMode, 418
local ref mkEvalable, 903
local ref retract, 1049
usedby displayValue, 455
usedby interpret2, 55
usedby recordAndPrint, 56
$EndOfOutput
usedby serverReadLine, 45
$EndServerSession, 43
usedby serverReadLine, 45
defvar, 43
$EndSession
usedby serverReadLine, 45
$FormalMapVariableList
local ref domArg, 420
local ref mkDomPvar, 431
local ref replaceSharps, 948
usedby displayOperationsFromLisplib, 813
usedby localnrlib, 1007
usedby reportOpsFromLisplib, 810
$FunctionalExpression
local ref defaultTargetFE, 435
$HTCompanionWindowID, 52
usedby recordAndPrint, 56
defvar, 52
$HiFiAccess, 724
usedby createCurrentInterpreterFrame, 553
usedby disableHist, 599
usedby emptyInterpreterFrame, 552
usedby historySpad2Cmd, 579
usedby initHist, 577
usedby putHist, 587
usedby restoreHistory, 593
usedby saveHistory, 592
usedby setHistoryCore, 581

usedby undoFromFile, 590
usedby undoInCore, 589
usedby updateFromCurrentInterpreterFrame, 554
usedby updateHist, 586
usedby writeInputLines, 583
defvar, 724
$HistListAct
usedby changeHistListLen, 585
usedby createCurrentInterpreterFrame, 553
usedby emptyInterpreterFrame, 552
usedby initHistList, 577
usedby resetInCoreHist, 584
usedby updateFromCurrentInterpreterFrame, 554
usedby updateInCoreHist, 586
$HistListLen
usedby changeHistListLen, 585
usedby createCurrentInterpreterFrame, 553
usedby emptyInterpreterFrame, 552
usedby initHistList, 577
usedby resetInCoreHist, 584
usedby undoInCore, 589
usedby updateFromCurrentInterpreterFrame, 554
usedby updateInCoreHist, 586
$HistList
usedby changeHistListLen, 585
usedby createCurrentInterpreterFrame, 553
usedby emptyInterpreterFrame, 552
usedby initHistList, 577
usedby recordOldValue0, 588
usedby resetInCoreHist, 584
usedby undoChanges, 589
usedby undoInCore, 589
usedby updateFromCurrentInterpreterFrame, 554
usedby updateInCoreHist, 586
$HistRecord
usedby createCurrentInterpreterFrame, 553
usedby emptyInterpreterFrame, 552
usedby initHistList, 577
usedby recordNewValue0, 587
usedby updateFromCurrentInterpreterFrame, 554
usedby updateHist, 586
usedby writeHiFi, 598
$IOindex
usedby createCurrentInterpreterFrame, 553
usedby historySpad2Cmd, 579
usedby mkprompt, 42
usedby removeUndoLines, 924
usedby resetWorkspaceVariables, 646
usedby restart, 18
usedby setHistoryCore, 581
usedby setIOindex, 595
usedby undoCount, 919
usedby undoInCore, 589
usedby undoSteps, 920
usedby updateFromCurrentInterpreterFrame, 554
usedby updateHist, 585
usedby writeHiFi, 598
usedby writeInputLines, 583
$InitialCommandSynonymAlist, 472
usedby resetWorkspaceVariables, 646
defvar, 472
$InitialModemapFrame, 9
usedby makeInitialModemapFrame, 37
defvar, 9
$Integer
local ref defaultTargetFE, 435
local ref hasCateSpecialNew, 433
local ref hasCateSpecial, 432
local ref mkEvalable, 903
$InteractiveFrame
local ref isDomainValuedVariable, 949
usedby ?t, 893
usedby augmentTraceNames, 862
usedby clearCmdAll, 499
usedby clearCmdParts, 501
usedby createCurrentInterpreterFrame, 553
usedby getAliasIfTracedMapParameter, 880
usedby getBpiNameIfTracedMap, 880
usedby getMapSig, 843
usedby getMapSubNames, 859
usedby getWorkspaceNames, 451
usedby interpFunctionDepAlists, 461
usedby isInterpOnlyMap, 862
usedby isUncompiledMap, 861
usedby ncTopLevel, 25

usedby parseAndInterpret, 48
usedby processInteractive1, 52
usedby recordFrame, 913
usedby reportUndo, 918
usedby restart, 18
usedby restoreHistory, 593
usedby showSpad2Cmd, 807
usedby undoChanges, 590
usedby undoFromFile, 590
usedby undoInCore, 589
usedby undoSteps, 920
usedby undo, 912
usedby updateFromCurrentInterpreterFrame, 554
usedby writeHistModesAndValues, 599
$InteractiveMode, 24
local ref loadLibNoUpdate, 1031
local ref loadLib, 1030
usedby addBinding, 953
usedby ncTopLevel, 25
usedby parseAndInterpret, 48
usedby readSpad2Cmd, 638
usedby zsystemDevelopmentSpad2Cmd, 943
usedby zsystemdevelopment1, 944
defvar, 24
$JoinOfCatDatabase
usedby clearCmdCompletely, 498
$JoinOfDomDatabase
usedby clearCmdCompletely, 498
$KillLispSystem
usedby serverReadLine, 45
$LispCommand
usedby serverReadLine, 45
$MenuServer
usedby executeQuietCommand, 47
usedby serverReadLine, 45
$NeedToSignalSessionManager, 44
usedby intloopSpadProcess, 65
usedby serverReadLine, 45
defvar, 44
$NonNullStream, 607
usedby dewritify,dewritifyInner, 608
usedby writify,writifyInner, 603
defvar, 607
$NonSmanSession
usedby serverReadLine, 45
$NullStream, 607
usedby dewritify,dewritifyInner, 608
usedby writify,writifyInner, 603
defvar, 607
$OutputForm
usedby coerceSpadArgs2E, 855
usedby coerceSpadFunValue2E, 858
usedby coerceTraceArgs2E, 854
usedby coerceTraceFunValue2E, 857
$PrintCompilerMessageIfTrue
usedby spad, 20
$ProcessInteractiveValue, 52
usedby processInteractive1, 52
usedby processInteractive, 50
defvar, 52
$QueryClients
usedby queryClients, 506
$QuickLet
usedby breaklet, 895
usedby tracelet, 894
$QuietCommand, 47
usedby executeQuietCommand, 47
usedby pf2Sex1, 300
usedby pf2Sex, 297
usedby recordAndPrint, 56
defvar, 47
$QuietSpadCommand
usedby serverReadLine, 45
$QuotientField
local ref hasCateSpecial, 432
$RTspecialCharacters, 968
usedby setOutputCharacters, 758
defvar, 968
$RationalNumber
local ref defaultTargetFE, 435
local ref hasCateSpecialNew, 433
$SessionManager
usedby close, 507
usedby queryClients, 506
usedby serverReadLine, 45
$SpadCommand
usedby serverReadLine, 45
$SpadServerName, 12, 13
defvar, 12, 13
$SpadServername

usedby restart, 18
$SpadServer, 12
usedby close, 507
usedby restart, 18
usedby serverReadLine, 45
usedby spad-save, 980
defvar, 12
$StreamFrame
usedby processInteractive, 50
$Subst
local def ofCategory, 417
local ref hasCate, 431
local ref unifyStructVar, 427
$SwitchFrames
usedby serverReadLine, 45
$Symbol
local ref defaultTargetFE, 435
$ThrowAwayMode
usedby interpret2, 55
$TriangleVariableList
local ref replaceSharps, 948
$UserAbbreviationsAlist
usedby resetWorkspaceVariables, 646
$UserLevel, 798
usedby getDirectoryList, 974
usedby readSpad2Cmd, 638
usedby satisfiesUserLevel, 447
usedby set1, 801
usedby synonymsForUserLevel, 825
usedby userLevelErrorMessage, 447
usedby whatCommands, 932
defvar, 798
$Void
usedby clearCmdSortedCaches, 497
usedby recordAndPrint, 56
$abbreviateTypes, 752
defvar, 752
$algebraFormat, 753
usedby setOutputAlgebra, 754
defvar, 753
$algebraOutputFile, 753
usedby setOutputAlgebra, 754
defvar, 753
$algebraOutputStream, 754
usedby recordAndPrint, 56
usedby sayMSG, 329
usedby setOutputAlgebra, 754
defvar, 754
$analyzingMapList
usedby processInteractive, 50
$ans
usedby nplisp, 468
$attrCats, 362
usedby whichCat, 363
defvar, 362
$attributeDb
usedby clearCmdCompletely, 498
$boot, 25
usedby ioclear, 962
usedby iostat, 960
usedby ncTopLevel, 25
usedby parseAndInterpret, 48
usedby resetWorkspaceVariables, 646
defvar, 25
$cacheAlist, 699
defvar, 699
$cacheMessages, 324
defvar, 324
$clearExcept, 495
usedby clearSpad2Cmd, 496
defvar, 495
$clearOptions, 495
usedby clearCmdExcept, 501
usedby clearCmdParts, 501
usedby clearSpad2Cmd, 496
defvar, 495
$coerceIntByMapCounter
usedby resetWorkspaceVariables, 646
$collectOutput
usedby printStatisticsSummary, 57
usedby printStorage, 58
usedby printTypeAndTimeNormal, 59
usedby recordAndPrint, 56
$commentedOps
usedby reportOpsFromUnitDirectly, 814
$compErrorMessageStack
usedby processInteractive, 50
$compileDontDefineFunctions, 703
defvar, 703
$compileMapFlag
usedby resetWorkspaceVariables, 646
$compileRecurrence, 704

defvar, 704
$compilingLoop
usedby processInteractive, 50
$compilingMap
usedby processInteractive, 50
$constructorList
usedby make-databases, 1010
$constructors, 889
usedby addTraceItem, 892
usedby traceReply, 889
defvar, 889
$current-directory, 7
usedby getDirectoryList, 974
usedby reroot, 41
usedby restart, 18
defvar, 7
$current-line
usedby iostat, 960
usedby spad-short-error, 960
$currentCarrier
usedby intloopSpadProcess, 65
$currentFrameNum, 43
usedby close, 507
usedby serverReadLine, 45
defvar, 43
$currentLine
usedby ExecuteInterpSystemCommand, 33
usedby clearCmdAll, 500
usedby getSystemCommandLine, 826
usedby intnplisp, 36
usedby removeUndoLines, 924
usedby restart, 18
usedby setCurrentLine, 42
usedby unAbbreviateKeyword, 465
usedby updateHist, 586
usedby writeHiFi, 598
$dalymode, 655
usedby intloopReadConsole, 31
defvar, 655
$declaredMode
usedby processInteractive, 50
$defaultFortVar
usedby processInteractive, 50
$defaultFortranType, 709
defvar, 709
$defaultMsgDatabaseName, 8
usedby fetchKeyedMsg, 326
usedby reroot, 41
defvar, 8
$defaultSpecialCharacters, 965
defvar, 965
$depTb
local ref saveDependentsHashTable, 1013
$dependeeAlist
usedby displayProperties,sayFunctionDeps, 454
usedby displayProperties, 458
usedby interpFunctionDepAlists, 461
$dependeeClosureAlist
usedby resetWorkspaceVariables, 646
$dependentAlist
usedby displayProperties,sayFunctionDeps, 454
usedby displayProperties, 458
usedby interpFunctionDepAlists, 461
$describeOptions, 524
usedby describeSpad2Cmd, 525
defvar, 524
$directory-list, 8
usedby getDirectoryList, 974
usedby reroot, 41
defvar, 8
$displayDroppedMap, 730
defvar, 730
$displayMsgNumber, 737
usedby sayKeyedMsgLocal, 328
defvar, 737
$displayOptions, 531
usedby displaySpad2Cmd, 532
defvar, 531
$displaySetValue, 739
usedby set1, 801
defvar, 739
$displayStartMsgs, 740
usedby restart, 18
defvar, 740
$doNotAddEmptyModeIfTrue
usedby domainToGenvar, 851
usedby reportOperations, 808
usedby transTraceItem, 853
$domPvar, 49

local def hasAtt, 422
local def hasCate1, 425
local def hasCaty1, 429
local def hasSig, 421
local ref domArg2, 420
local ref hasCaty, 419
local ref unifyStructVar, 427
usedby processInteractive, 50
defvar, 49
$domainTraceNameAssoc
usedby genDomainTraceName, 852
$domains
usedby addTraceItem, 892
usedby traceReply, 889
$dotdot
usedby opTran, 321
$echoLineStack
usedby resetWorkspaceVariables, 646
$envHashTable, 953
usedby addBinding, 953
defvar, 953
$env
local ref isDomainValuedVariable, 949
usedby interpret, 54
usedby reportOperations, 808
usedby resetWorkspaceVariables, 646
usedby showSpad2Cmd, 807
$erMsgToss
usedby SpadInterpretStream, 29
usedby initImPr, 365
usedby phIntReportMsgs, 66
usedby showMsgPos?, 356
$erase
local ref saveDependentsHashTable, 1013
local ref saveUsersHashTable, 1014
usedby deleteFile, 1037
usedby reportOpsFromLisplib1, 809
usedby reportOpsFromUnitDirectly1, 818
$evalDomain
local ref evalDomain, 903
$eval
usedby interpret1, 54
usedby interpret, 54
usedby reportOperations, 808
$existingFiles
usedby clearCmdCompletely, 499
usedby resetWorkspaceVariables, 646
$expandSegments
local def evaluateType, 906
$e
local ref isDomainValuedVariable, 949
usedby clearCmdParts, 501
usedby describeSpad2Cmd, 525
usedby displaySpad2Cmd, 532
usedby interpFunctionDepAlists, 461
usedby ncTopLevel, 25
usedby parseAndInterpret, 48
usedby processInteractive1, 52
usedby recordAndPrint, 56
usedby resetWorkspaceVariables, 646
usedby restoreHistory, 593
usedby showSpad2Cmd, 807
usedby systemCommand, 444
usedby whatSpad2Cmd, 930
$filep
usedby setOutputAlgebra, 754
usedby setOutputFormula, 782
usedby setOutputFortran, 761
usedby setOutputHtml, 773
usedby setOutputMathml, 768
usedby setOutputOpenMath, 777
usedby setOutputTex, 788
$findfile
usedby readSpad2Cmd, 638
$floatok
usedby lineoftoks, 109
usedby scanKeyTr, 117
usedby scanNumber, 129
usedby scanSpace, 127
usedby scanString, 127
usedby scanWord, 123
$fn
usedby SpadInterpretStream, 29
usedby toFile?, 361
$forceDatabaseUpdate
local ref loadLib, 1030
local ref updateDatabase, 1009
usedby localdatabase, 1005
$formulaFormat, 781
usedby setOutputFormula, 782
defvar, 781
$formulaOutputFile, 781

usedby setOutputFormula, 782
defvar, 781
$formulaOutputStream
usedby setOutputFormula, 782
$fortIndent, 706
defvar, 706
$fortInts2Floats, 706
defvar, 706
$fortLength, 707
defvar, 707
$fortPersistence, 747
usedby describeFortPersistence, 748
usedby setFortPers, 748
defvar, 747
$fortVar
usedby processInteractive, 50
$fortranArrayStartingIndex, 713
defvar, 713
$fortranDirectory, 717
usedby describeSetFortDir, 718
usedby setFortDir, 718
defvar, 717
$fortranFormat, 760
usedby setOutputFortran, 761
defvar, 760
$fortranLibraries, 719
usedby describeSetLinkerArgs, 720
usedby setLinkerArgs, 720
defvar, 719
$fortranOptimizationLevel, 712
defvar, 712
$fortranOutputFile, 760
usedby setOutputFortran, 761
defvar, 760
$fortranOutputStream
usedby setOutputFortran, 761
$fortranPrecision, 709
defvar, 709
$fortranSegment, 712
defvar, 712
$fortranTmpDir, 714
usedby describeSetFortTmpDir, 716
usedby setFortTmpDir, 715
defvar, 714
$fractionDisplayType, 765
defvar, 765
$frameAlist, 43
usedby serverReadLine, 45
defvar, 43
$frameMessages, 732
usedby clearCmdAll, 500
usedby displayProperties, 458
usedby updateFromCurrentInterpreterFrame, 554
defvar, 732
$frameNumber, 43
usedby serverReadLine, 45
defvar, 43
$frameRecord, 911
usedby clearCmdAll, 499
usedby clearFrame, 919
usedby recordFrame, 913
usedby undoSteps, 920
defvar, 911
$freeVars
usedby processInteractive, 50
$fromSpadTrace
usedby spadTrace, 867
$fullScreenSysVars, 721
defvar, 721
$functionTable, 498
usedby clearCmdCompletely, 499
usedby resetWorkspaceVariables, 646
defvar, 498
$funnyBacks
usedby unescapeStringsInForm, 62
$funnyQuote
usedby unescapeStringsInForm, 62
$f
usedby lineoftoks, 109
usedby nextline, 110
$genValue, 53
usedby interpret1, 54
usedby interpret, 54
usedby reportOperations, 808
defvar, 53
$giveExposureWarning, 730
defvar, 730
$globalExposureGroupAlist, 662
local ref isExposedConstructor, 812
usedby setExposeAddGroup, 691
usedby setExposeDropGroup, 694

defvar, 662
$hashOp0
local ref basicLookup, 1061
$hashOp1
local ref basicLookup, 1061
$hashOpApply
local ref basicLookup, 1061
$hashOpSet
local ref basicLookup, 1061
$hashSeg
local ref basicLookup, 1061
$highlightAllowed, 733
defvar, 733
$historyDirectory, 575
usedby histInputFileName, 576
defvar, 575
$historyDisplayWidth, 722
defvar, 722
$historyFileType, 575
defvar, 575
$hope
local def hasCate, 431
local def ofCategory, 417
local def unifyStructVar, 427
$htmlFormat, 771
usedby setOutputHtml, 773
defvar, 771
$htmlOutputFile, 771
usedby setOutputHtml, 773
defvar, 771
$htmlOutputStream
usedby setOutputHtml, 772
$imPrGuys, 356
defvar, 356
$imPrTagGuys, 365
usedby initImPr, 365
defvar, 365
$inRetract
usedby processInteractive, 50
$inclAssertions
usedby SpadInterpretStream, 29
usedby assertCond, 96
usedby ifCond, 89
$includeUnexposed?
usedby getBrowseDatabase, 1075
$inputPromptType, 738
usedby mkprompt, 42
defvar, 738
$inputStream
usedby npFirstTok, 141
usedby npListAndRecover, 187
usedby npMoveTo, 189
usedby npNext, 143
usedby npParse, 139
usedby npRestore, 150
usedby npState, 210
$insideApplication
usedby pf2Sex, 297
usedby pfApplication2Sex, 304
usedby pfDefinition2Sex, 313
$insideRule
usedby pf2Sex1, 300
usedby pf2Sex, 297
usedby pfApplication2Sex, 304
usedby pfLhsRule2Sex, 316
usedby pfLiteral2Sex, 302
usedby pfOp2Sex, 306
usedby pfRhsRule2Sex, 317
$insideSEQ
usedby pf2Sex1, 300
usedby pf2Sex, 297
usedby pfSequence2Sex, 308
$instantCanCoerceCount
usedby processInteractive, 50
$instantCoerceCount
usedby processInteractive, 50
$instantMmCondCount
usedby processInteractive, 50
$instantRecord
usedby processInteractive, 50
$intCoerceFailure, 32
usedby InterpExecuteSpadSystemCommand, 32
usedby intloopSpadProcess, 65
defvar, 32
$intRestart, 26
usedby intloop, 26
defvar, 26
$intSpadReader, 32
usedby InterpExecuteSpadSystemCommand, 32
usedby intloopSpadProcess, 65

defvar, 32
$intTopLevel, 26
usedby intloop, 26
defvar, 26
$internalHistoryTable
usedby clearCmdAll, 499
usedby createCurrentInterpreterFrame, 553
usedby readHiFi, 597
usedby restoreHistory, 593
usedby saveHistory, 592
usedby setHistoryCore, 581
usedby updateFromCurrentInterpreterFrame, 554
usedby writeHiFi, 598
$interpOnly, 49
usedby processInteractive, 50
defvar, 49
$interpreterFrameName
usedby clearCmdAll, 500
usedby closeInterpreterFrame, 558
usedby createCurrentInterpreterFrame, 553
usedby displayExposedGroups, 696
usedby displayProperties, 458
usedby histFileName, 576
usedby histInputFileName, 576
usedby initializeInterpreterFrameRing, 551
usedby mkprompt, 42
usedby setExposeAddConstr, 692
usedby setExposeAddGroup, 691
usedby setExposeDropConstr, 695
usedby setExposeDropGroup, 694
usedby updateFromCurrentInterpreterFrame, 554
$interpreterFrameRing
usedby addNewInterpreterFrame, 557
usedby changeToNamedInterpreterFrame, 556
usedby closeInterpreterFrame, 558
usedby displayFrameNames, 559
usedby findFrameInRing, 555
usedby frameNames, 552
usedby importFromFrame, 559
usedby initializeInterpreterFrameRing, 551
usedby nextInterpreterFrame, 556
usedby previousInterpreterFrame, 557
usedby updateCurrentInterpreterFrame, 555
usedby updateFromCurrentInterpreterFrame, 554
$interpreterTimedClasses
usedby printStorage, 58
usedby printTypeAndTimeNormal, 59
usedby printTypeAndTimeSaturn, 60
usedby processInteractive, 50
$interpreterTimedNames
usedby printStorage, 58
usedby printTypeAndTimeNormal, 59
usedby printTypeAndTimeSaturn, 60
usedby processInteractive, 50
$lambdatype, 654
defvar, 654
$lastLineInSEQ
usedby processInteractive, 50
$lastPos
usedby SpadInterpretStream, 29
usedby getPosStL, 354
$lastUntraced
usedby getMapSubNames, 859
usedby trace1, 838
usedby untraceMapSubNames, 863
usedby untrace, 852
$letAssoc
usedby breaklet, 895
usedby letPrint2, 877
usedby letPrint3, 878
usedby letPrint, 875
usedby spadTrace, 867
usedby spadUntrace, 886
usedby tracelet, 894
$libQuiet
usedby SpadInterpretStream, 29
$library-directory-list, 9
usedby getDirectoryList, 974
usedby reroot, 41
defvar, 9
$linearFormatScripts, 785
defvar, 785
$linelength, 765
usedby displayOperationsFromLisplib, 813
usedby displaySetOptionInformation, 647
usedby displaySetVariableSettings, 649

usedby filterAndFormatConstructors, 934
usedby justifyMyType, 62
usedby msgOutputter, 351
usedby msgText, 62
usedby printSynonyms, 470
usedby reportOpsFromLisplib, 810
usedby reportOpsFromUnitDirectly, 814
usedby sayKeyedMsgLocal, 328
usedby setExposeAddConstr, 692
usedby setExposeAddGroup, 691
usedby setExposeAdd, 690
usedby setExposeDropConstr, 695
usedby setExposeDropGroup, 694
usedby setExposeDrop, 693
usedby spadStartUpMsgs, 19
usedby traceReply, 889
usedby whatCommands, 932
usedby workfilesSpad2Cmd, 940
defvar, 765

$linepos
usedby lineoftoks, 109
usedby nextline, 110
usedby scanCheckRadix, 132
usedby scanError, 133
usedby scanS, 128
usedby scanToken, 112

$lines
usedby intloopEchoParse, 69
usedby intloopInclude0, 63
usedby ncloopInclude0, 73

$ln
usedby lineoftoks, 109
usedby nextline, 110
usedby scanComment, 113
usedby scanError, 133
usedby scanEsc, 121
usedby scanExponent, 124
usedby scanNegComment, 114
usedby scanNumber, 130
usedby scanPossFloat, 118
usedby scanPunct, 115
usedby scanSpace, 127
usedby scanS, 128
usedby scanToken, 112
usedby scanW, 125
usedby spleI1, 120
usedby startsComment?, 113
usedby startsNegComment?, 114

$localExposureDataDefault, 688
usedby clearCmdCompletely, 499
usedby emptyInterpreterFrame, 552
defvar, 688

$localExposureData, 688
local ref isExposedConstructor, 812
usedby clearCmdCompletely, 498
usedby createCurrentInterpreterFrame, 553
usedby displayExposedConstructors, 696
usedby displayExposedGroups, 696
usedby displayHiddenConstructors, 697
usedby setExposeAddConstr, 692
usedby setExposeAddGroup, 691
usedby setExposeDropConstr, 695
usedby setExposeDropGroup, 694
usedby updateFromCurrentInterpreterFrame, 554
defvar, 688

$localVars
usedby processInteractive, 50

$lookupDefaults
local def oldCompLookup, 1064
local ref basicLookupCheckDefaults, 1063
usedby clearCmdSortedCaches, 497

$macActive
usedby mac0ExpandBody, 223
usedby mac0MLambdaApply, 222
usedby macId, 225
usedby macroExpanded, 220

$mapList
usedby processInteractive, 50

$mapSubNameAlist
usedby ?t, 893
usedby isSubForRedundantMapName, 863
usedby saveMapSig, 843
usedby traceSpad2Cmd, 837
usedby untraceMapSubNames, 863
usedby untrace, 852

$margin, 765
usedby msgText, 62
usedby sayKeyedMsgLocal, 328
defvar, 765

$mathTraceList
usedby coerceTraceArgs2E, 854

usedby coerceTraceFunValue2E, 857
$mathmlFormat, 767
usedby setOutputMathml, 768
defvar, 767
$mathmlOutputFile, 767
usedby setOutputMathml, 768
defvar, 767
$mathmlOutputStream
usedby setOutputMathml, 768
$maximumFortranExpressionLength, 711
defvar, 711
$minivectorCode
usedby processInteractive, 50
$minivectorNames, 49
usedby processInteractive, 50
defvar, 49
$minivector
usedby processInteractive, 50
$mkTestFlag
usedby recordAndPrint, 56
$mkTestInputStack
usedby updateHist, 586
$mkTestOutputType
usedby recordAndPrint, 56
$msgAlist, 324
usedby resetWorkspaceVariables, 646
usedby spadStartUpMsgs, 19
defvar, 324
$msgDatabaseName, 9, 10, 324
usedby getMsgInfoFromKey, 364
usedby reroot, 41
usedby resetWorkspaceVariables, 646
defvar, 9, 10, 324
$msgDatabase
usedby resetWorkspaceVariables, 646
$msgdbNoBlanksAfterGroup, 326
defvar, 326
$msgdbNoBlanksBeforeGroup, 326
defvar, 326
$msgdbPrims, 325
usedby fixObjectForPrinting, 452
defvar, 325
$msgdbPunct, 325
defvar, 325
$multiVarPredicateList
usedby pfRule2Sex, 316
usedby ruleLhsTran, 320
usedby rulePredicateTran, 317
$nagEnforceDouble, 750
defvar, 750
$nagHost, 745
usedby describeSetNagHost, 746
usedby setNagHost, 746
defvar, 745
$nagMessages, 736
defvar, 736
$ncMsgList, 27
usedby SpadInterpretStream, 29
usedby intloopSpadProcess, 65
usedby ncConversationPhase,wrapup, 68
usedby ncConversationPhase, 68
usedby processKeyedError, 351
defvar, 27
$newConlist
usedby library, 1005
$newcompErrorCount, 28
usedby SpadInterpretStream, 29
usedby ncBug, 366
usedby ncHardError, 350
usedby ncSoftError, 349
defvar, 28
$newspad
usedby ncTopLevel, 25
$noEvalTypeMsg, 909
local ref throwEvalTypeMsg, 909
defvar, 909
$noParseCommands, 440
usedby doSystemCommand, 442
defvar, 440
$noRepList
usedby processMsgList, 367
usedby redundant, 372
$noSubsumption, 952
defvar, 952
$nopos, 28
usedby SpadInterpretStream, 29
usedby makeLeaderMsg, 375
usedby ncBug, 366
usedby pfSourcePosition, 236
usedby poNoPosition, 412
defvar, 28
$npPParg

usedby npPPff, 208
$npTokToNames
usedby npId, 203
$numericFailure, 1065
defvar, 1065
$n
usedby lineoftoks, 109
usedby nextline, 110
usedby scanCheckRadix, 132
usedby scanComment, 113
usedby scanError, 133
usedby scanEscape, 132
usedby scanEsc, 121
usedby scanExponent, 124
usedby scanNegComment, 115
usedby scanNumber, 130
usedby scanPossFloat, 119
usedby scanPunct, 115
usedby scanSpace, 127
usedby scanString, 127
usedby scanS, 128
usedby scanToken, 112
usedby scanW, 125
usedby spleI1, 120
usedby startsComment?, 113
usedby startsNegComment?, 114
$okToExecuteMachineCode
usedby SpadInterpretStream, 29
$oldBreakMode, 1065
defvar, 1065
$oldHistoryFileName, 574
usedby oldHistFileName, 576
usedby restoreHistory, 593
defvar, 574
$oldline, 445
usedby commandErrorIfAmbiguous, 466
usedby commandErrorMessage, 446
defvar, 445
$opSysName
usedby spadStartUpMsgs, 19
$openMathFormat, 776
usedby setOutputOpenMath, 777
defvar, 776
$openMathOutputFile, 776
usedby setOutputOpenMath, 777
defvar, 776
$openMathOutputStream
usedby setOutputOpenMath, 777
$openServerIfTrue, 10
usedby restart, 18
usedby spad-save, 980
defvar, 10
$operationNameList
usedby resetWorkspaceVariables, 646
$optionAlist, 836
usedby trace1, 839
defvar, 836
$options
usedby abbreviationsSpad2Cmd, 479
usedby clearSpad2Cmd, 496
usedby close, 507
usedby frameSpad2Cmd, 562
usedby historySpad2Cmd, 579
usedby history, 578
usedby library, 1005
usedby readSpad2Cmd, 638
usedby reportOpsFromLisplib, 810
usedby reportOpsFromUnitDirectly, 814
usedby restoreHistory, 593
usedby showSpad2Cmd, 807
usedby systemCommand, 444
usedby trace1, 838
usedby undo, 912
usedby workfilesSpad2Cmd, 940
usedby zsystemdevelopment1, 944
$op
usedby displayMacro, 450
usedby displayType, 456
usedby displayValue, 455
usedby processInteractive, 50
$outputLibraryName
usedby setOutputLibrary, 656
$outputLines
usedby printTypeAndTimeNormal, 59
$outputList
usedby processMsgList, 367
usedby queueUpErrors, 370
$outputMode
usedby recordAndPrint, 56, 57
$packages
usedby addTraceItem, 892
usedby traceReply, 889

$pfMacros, 98
usedby clearMacroTable, 500
usedby clearParserMacro, 449
usedby displayParserMacro, 460
usedby getParserMacroNames, 449
usedby mac0Define, 228
usedby mac0GetName, 224
usedby mac0Get, 226
usedby mac0MLambdaApply, 222
usedby macApplication, 221
usedby macLambda,mac, 227
usedby macLambda, 227
usedby macWhere,mac, 226
usedby macWhere, 226
defvar, 98
$plainRTspecialCharacters, 968
usedby setOutputCharacters, 758
defvar, 968
$plainSpecialCharacters0, 966
defvar, 966
$plainSpecialCharacters1, 966
defvar, 966
$plainSpecialCharacters2, 967
defvar, 967
$plainSpecialCharacters3, 967
defvar, 967
$posActive
usedby mac0ExpandBody, 223
usedby mac0MLambdaApply, 222
usedby macId, 225
usedby macroExpanded, 220
$preLength, 353
usedby getPreStL, 353
usedby makeLeaderMsg, 375
usedby makeMsgFromLine, 369
usedby processChPosesForOneLine, 374
usedby tabbing, 360
defvar, 353
$predicateList
usedby pfRule2Sex, 316
usedby pfSuchThat2Sex, 305
usedby ruleLhsTran, 320
$prettyprint, 797
defvar, 797
$prevCarrier
usedby intloopSpadProcess, 65
$previousBindings, 911
usedby clearCmdAll, 499
usedby clearFrame, 919
usedby recordFrame, 913
defvar, 911
$printAnyIfTrue, 726
usedby recordAndPrint, 57
defvar, 726
$printFortranDecs, 708
defvar, 708
$printLoadMsgs, 727
local ref loadLibNoUpdate, 1031
local ref loadLib, 1030
usedby restart, 18
defvar, 727
$printMsgsToFile, 731
usedby sayKeyedMsgLocal, 328
defvar, 731
$printStatisticsSummaryIfTrue, 741
usedby recordAndPrint, 56
defvar, 741
$printStorageIfTrue
usedby recordAndPrint, 56
$printTimeIfTrue, 742
usedby printTypeAndTimeNormal, 59
usedby printTypeAndTimeSaturn, 60
usedby recordAndPrint, 56
defvar, 742
$printTypeIfTrue, 743
usedby printTypeAndTimeNormal, 59
usedby printTypeAndTimeSaturn, 60
usedby recordAndPrint, 56
defvar, 743
$printVoidIfTrue, 744
usedby recordAndPrint, 56
defvar, 744
$promptMsg, 28
usedby SpadInterpretStream, 29
defvar, 28
$quadSymbol
local ref evaluateType1, 908
usedby reportOperations, 808
$quitCommandType, 791
usedby pquitSpad2Cmd, 630
usedby quitSpad2Cmd, 634
defvar, 791

$quitTag, 20
  usedby runspad, 21
  defvar, 20
$quotedOpList
  usedby pfOp2Sex, 306
  usedby pfRule2Sex, 316
$relative-directory-list, 10, 11
  usedby reroot, 41
  defvar, 10, 11
$relative-library-directory-list, 11
  usedby reroot, 41
  defvar, 11
$repGuys, 373
  defvar, 373
$reportBottomUpFlag, 728
  defvar, 728
$reportCoerceIfTrue, 729
  defvar, 729
$reportCompilation, 795
  defvar, 795
$reportInstantiations, 734
  usedby processInteractive, 50
  defvar, 734
$reportInterpOnly, 735
  defvar, 735
$reportOptimization, 796
  defvar, 796
$reportSpadTrace, 836
  usedby spadTrace, 867
  defvar, 836
$reportUndo, 912
  defvar, 912
$runTestFlag
  usedby recordAndPrint, 56
$r
  usedby lineoftoks, 109
  usedby nextline, 110
  usedby scanEsc, 121
$saturn
  usedby printTypeAndTime, 58
$sayBrightlyStream
  usedby reportOpsFromLisplib1, 809
  usedby reportOpsFromUnitDirectly1, 817
$seen
  usedby ScanOrPairVec,ScanOrInner, 611
  usedby ScanOrPairVec, 612
  usedby dewritify,dewritifyInner, 608
  usedby dewritify, 611
  usedby saveHistory, 592
  usedby writify,writifyInner, 603
  usedby writify, 606
$setOptionNames, 799
  usedby set1, 801
  defvar, 799
$setOptions
  usedby resetWorkspaceVariables, 646
  usedby set, 799
$showOptions
  usedby reportOpsFromLisplib, 810
  usedby reportOpsFromUnitDirectly, 814
  usedby showSpad2Cmd, 807
$slamFlag
  usedby resetWorkspaceVariables, 646
$sockBufferLength, 44
  usedby serverReadLine, 45
  defvar, 44
$sourceFiles
  usedby resetWorkspaceVariables, 646
  usedby updateSourceFiles, 542
  usedby workfilesSpad2Cmd, 940
$spad-errors, 957
  usedby init-boot/spad-reader, 958
  defvar, 957
$spadroot, 11, 12
  local ref isSystemDirectory, 1030
  usedby DaaseName, 1014
  usedby browseOpen, 996
  usedby categoryOpen, 997
  usedby compressOpen, 1015
  usedby getdatabase, 1001
  usedby initroot, 35
  usedby interpOpen, 995
  usedby make-absolute-filename, 36
  usedby operationOpen, 998
  usedby reroot, 41
  usedby write-browsedb, 1024
  usedby write-interpdb, 1022
  defvar, 11, 12
$spad
  usedby ioclear, 962
  usedby iostat, 960
  usedby ncTopLevel, 25

usedby parseAndInterpret, 48
$specialCharacterAlist, 969
usedby setOutputCharacters, 758
usedby specialChar, 970
defvar, 969
$specialCharacters, 969
usedby setOutputCharacters, 758
usedby specialChar, 970
defvar, 969
$stack
usedby npListAndRecover, 187
usedby npListofFun, 219
usedby npList, 153
usedby npParse, 139
usedby npPop1, 142
usedby npPop2, 142
usedby npPop3, 142
usedby npPushId, 207
usedby npPush, 141
usedby npRestore, 150
usedby npState, 210
usedby npZeroOrMore, 175
$stepNo
usedby intloopSpadProcess, 65
$stok
usedby npConstTok, 182
usedby npDDInfKey, 206
usedby npDollar, 182
usedby npEnclosed, 209
usedby npEqKey, 143
usedby npEqPeek, 150
usedby npFirstTok, 141
usedby npId, 203
usedby npInfKey, 206
usedby npInfixOperator, 158
usedby npInfixOp, 159
usedby npMissing, 149
usedby npParenthesize, 213
usedby npParse, 139
usedby npPrefixColon, 159
usedby npPushId, 207
usedby npRecoverTrap, 188
usedby npTrap, 210
usedby sySpecificErrorHere, 190
$streamCount, 792
usedby coerceSpadArgs2E, 855
usedby coerceSpadFunValue2E, 858
usedby describeSetStreamsCalculate, 794
usedby setStreamsCalculate, 793
defvar, 792
$streamsShowAll, 794
defvar, 794
$syscommands, 440
usedby newHelpSpad2Cmd, 569
usedby systemCommand, 444
usedby unAbbreviateKeyword, 465
defvar, 440
$systemCommandFunction
usedby SpadInterpretStream, 29
usedby intloopProcess, 64
usedby ncloopCommand, 474
$systemCommands, 439
usedby synonymsForUserLevel, 825
usedby systemCommand, 444
usedby unAbbreviateKeyword, 465
usedby whatCommands, 932
defvar, 439
$sz
usedby lineoftoks, 109
usedby nextline, 110
usedby scanComment, 113
usedby scanEsc, 121
usedby scanExponent, 124
usedby scanNegComment, 114
usedby scanNumber, 130
usedby scanPossFloat, 118
usedby scanS, 128
usedby scanW, 125
usedby spleI1, 120
usedby startsComment?, 113
usedby startsNegComment?, 114
$testingErrorPrefix, 325
defvar, 325
$testingSystem, 742
defvar, 742
$texFormatting, 325
usedby sayKeyedMsg, 328
defvar, 325
$texFormat, 787
usedby setOutputTex, 788
defvar, 787
$texOutputFile, 787

usedby setOutputTex, 788
defvar, 787
$texOutputStream
usedby printAsTeX, 61
usedby setOutputTex, 788
$timeGlobalName
usedby processInteractive, 50
$timedNameStack
usedby interpretTopLevel, 53
$toWhereGuys, 361
defvar, 361
$tokenCommands, 471
usedby doSystemCommand, 442
defvar, 471
$topicHash
usedby write-warmdata, 1027
$traceErrorStack
usedby getTraceOptions, 842
usedby stackTraceOptionError, 851
$traceNoisely, 836
usedby reportSpadTrace, 882
usedby spadTrace, 867
usedby trace1, 838
defvar, 836
$traceOptionList, 836
usedby getTraceOption, 844
defvar, 836
$tracedMapSignatures, 836
usedby coerceTraceArgs2E, 854
usedby coerceTraceFunValue2E, 857
usedby removeTracedMapSigs, 854
usedby saveMapSig, 843
defvar, 836
$tracedModemap
usedby spadTrace, 867
$tracedSpadModemap
usedby coerceSpadArgs2E, 855
usedby coerceSpadFunValue2E, 858
$traceletFunctions
usedby breaklet, 895
usedby tracelet, 894
$traceletflag
usedby tracelet, 894
$ttok
usedby npEqKey, 143
usedby npEqPeek, 150
usedby npFirstTok, 141
usedby npId, 203
usedby npInfKey, 206
usedby npInfixOp, 159
usedby npParse, 139
usedby npPushId, 207
$underbar, 582
defvar, 582
$undoFlag, 911
usedby recordFrame, 913
defvar, 911
$useBFasDefault
usedby float2Sex, 303
$useEditorForShowOutput, 786
usedby reportOpsFromLisplib0, 809
usedby reportOpsFromUnitDirectly0, 813
defvar, 786
$useFullScreenHelp, 723
usedby newHelpSpad2Cmd, 569
defvar, 723
$useInternalHistoryTable, 575
usedby clearCmdAll, 499
usedby initHist, 577
usedby readHiFi, 597
usedby restoreHistory, 593
usedby saveHistory, 592
usedby setHistoryCore, 581
usedby writeHiFi, 598
defvar, 575
$useIntrinsicFunctions, 710
defvar, 710
$usersTb
local ref saveUsersHashTable, 1014
$variableNumberAlist
usedby clearCmdAll, 499
$whatOptions, 929
usedby reportWhatOptions, 931
usedby whatSpad2Cmd, 930
defvar, 929
$whereCacheList
usedby processInteractive, 50
$writifyComplained
usedby writifyComplain, 602
usedby writify, 606
$xdatabase
usedby clearCmdCompletely, 498

abbQuery, 532
  calledby abbreviationsSpad2Cmd, 479
  calledby displaySpad2Cmd, 531
  calls getdatabase, 532
  calls sayKeyedMsg, 532
  defun, 532
abbreviate
  calledby traceReply, 889
abbreviation?
  calledby abbreviationsSpad2Cmd, 479
abbreviations, 479
  calls abbreviationsSpad2Cmd, 479
  defun, 479
abbreviations help page, 477
  manpage, 477
abbreviationsSpad2Cmd, 479
  calledby abbreviations, 479
  calls abbQuery, 479
  calls abbreviation?, 479
  calls deldatabase, 479
  calls exit, 479
  calls helpSpad2Cmd, 479
  calls listConstructorAbbreviations, 479
  calls mkUserConstructorAbbreviation, 479
  calls opOf, 479
  calls qcar, 479
  calls qcdr, 479
  calls sayKeyedMsg, 479
  calls selectOptionLC, 479
  calls seq, 479
  calls setdatabase, 479
  calls size, 479
  uses $options, 479
  defun, 479
acot, 1089
  defun, 1089
acoth, 1092
  defun, 1092
acsc, 1090
  defun, 1090
acsch, 1091
  defun, 1091
addassoc
  calledby saveMapSig, 843
  calledby trace1, 838
addBinding, 953
  calls addBindingInteractive, 953
  calls getProplist, 953
  calls hput, 953
  uses $InteractiveMode, 953
  uses $envHashTable, 953
  defun, 953
addBindingInteractive, 957
  calledby addBinding, 953
  calls assq, 957
  defun, 957
addInputLibrary, 660
  calledby setInputLibrary, 659
  calls dropInputLibrary, 660
  uses input-libraries, 660
  defun, 660
addNewInterpreterFrame, 557
  calledby frameSpad2Cmd, 562
  calledby serverReadLine, 44
  calls $erase, 557
  calls boot-equal, 557
  calls emptyInterpreterFrame, 557
  calls framename, 557
  calls histFileName, 557
  calls initHistList, 557
  calls throwKeyedMsg, 557
  calls updateCurrentInterpreterFrame, 557
  calls updateFromCurrentInterpreterFrame, 557
  uses $interpreterFrameRing, 557
  defun, 557
addoperations, 998
  calledby localnrlib, 1007
  calls getdatabase, 998
  uses *operation-hash*, 999
  defun, 998
addTraceItem, 892
  calledby traceReply, 889
  calls constructor?, 892
  calls devaluate, 892
  calls isDomainOrPackage, 892
  calls isDomain, 892
  uses $constructors, 892
  uses $domains, 892
  uses $packages, 892
  defun, 892
aldorTrace

calledby spadTrace, 866
algCoerceInteractive, 1051
defun, 1051
allConstructors, 1027
calledby make-databases, 1010
calledby write-browsedb, 1024
calledby write-compress, 1016
uses *allconstructors*, 1027
defun, 1027
allocate
calledby init-memory-config, 34
allocate-contiguous-pages
calledby init-memory-config, 34
allocate-relocatable-pages
calledby init-memory-config, 34
allOperations, 1027
calledby apropos, 935
calledby write-compress, 1016
uses *allOperations*, 1027
uses *operation-hash*, 1028
defun, 1027
alqlGetKindString, 1074
calls dbPart, 1074
calls substring, 1074
defun, 1074
alqlGetOrigin, 1073
calls charPosition, 1073
calls dbPart, 1073
calls substring, 1073
defun, 1073
alqlGetParams, 1074
calls charPosition, 1074
calls dbPart, 1074
calls substring, 1074
defun, 1074
alreadyOpened?, 361
calledby msgOutputter, 351
calls msgImPr?, 361
defun, 361
apropos, 935
calledby whatSpad2Cmd, 930
calls allOperations, 935
calls downcase, 935
calls exit, 935
calls filterListOfStrings, 935
calls msort, 935
calls sayAsManyPerLineAsPossible, 935
calls sayKeyedMsg, 935
calls sayMessage, 935
calls seq, 935
defun, 935
as-insert
calledby spadTrace, 866
asec, 1090
defun, 1090
asech, 1092
defun, 1092
assertCond, 96
calledby incLude1, 82
calls MakeSymbol, 96
calls incCommandTail, 96
uses *whitespace*, 96
uses $inclAssertions, 96
defun, 96
assignment, 381
syntax, 381
assoc
calledby breaklet, 895
calledby clearCmdParts, 501
calledby clearParserMacro, 449
calledby diffAlist, 915
calledby getOption, 882
calledby readHiFi, 597
calledby spadTrace, 866
calledby spadUntrace, 885
assocleft
calledby clearCmdParts, 501
calledby isSubForRedundantMapName, 863
assocright
calledby orderBySlotNumber, 883
calledby untraceMapSubNames, 863
assq, 1044
calledby addBindingInteractive, 957
calledby diffAlist, 915
calledby fetchOutput, 596
calledby hasSig, 421
calledby recordNewValue0, 587
calledby recordOldValue0, 588
calledby searchCurrentEnv, 954
calledby searchTailEnv, 955
calledby showInOut, 596
calledby specialChar, 970

calledby undoFromFile, 590
calledby undoInCore, 589
calledby undoSingleStep, 921
defmacro, 1044
augmentSub
calledby hasCateSpecialNew, 433
calledby hasCateSpecial, 432
calledby hasCaty, 418
calledby unifyStructVar, 427
augmentTraceNames, 862
calledby traceSpad2Cmd, 837
calls get, 862
uses $InteractiveFrame, 862
defun, 862
AxiomServer
calledby browse, 489
axiomVersion, 472
uses *build-version*, 472
uses *yearweek*, 472
defun, 472
AXSERV;axServer;IMV;2
calledby browse, 489

basicLookup, 1061
calls HasCategory, 1061
calls error, 1061
calls hashCode?, 1061
calls hashString, 1061
calls hashType, 1061
calls isNewWorldDomain, 1061
calls lookupInDomainVector, 1061
calls oldCompLookup, 1061
calls opIsHasCat, 1061
calls spadcall, 1061
calls vecp, 1061
local ref $hashOp0, 1061
local ref $hashOp1, 1061
local ref $hashOpApply, 1061
local ref $hashOpSet, 1061
local ref $hashSeg, 1061
defun, 1061
basicLookupCheckDefaults, 1063
calledby lookupInDomainVector, 1063
calls error, 1063
calls hashCode?, 1063
calls hashString, 1063
calls hashType, 1063
calls spadcall, 1063
calls vecp, 1063
local ref $lookupDefaults, 1063
defun, 1063
basicMatch?
calledby stringMatches?, 1075
bit-to-truth, 1056
defmacro, 1056
blankList
calledby filterAndFormatConstructors, 934
calledby printLabelledList, 470
calledby whatCommands, 932
blocks, 384
syntax, 384
Boolean
calledby hashable, 1060
BooleanEquality, 1055
defun, 1055
boot help page, 483
manpage, 483
boot-equal
calledby addNewInterpreterFrame, 557
calledby clearCmdParts, 501
calledby displaySetOptionInformation, 647
calledby fetchOutput, 596
calledby findFrameInRing, 555
calledby getMapSig, 843
calledby importFromFrame, 559
calledby setHistoryCore, 581
calledby undoChanges, 589
calledby untraceDomainConstructor,keepTraced?, 872
calledby whatConstructors, 935
calledby writify,writifyInner, 603
boot-line-stack, 951
usedby init-boot/spad-reader, 958
usedby next-lines-clear, 962
usedby next-lines-show, 961
defvar, 951
bottomUp
calledby evaluateType, 906
calledby interpret1, 54
bottumUp
calledby evaluateType1, 908
bpiname

calledby breaklet, 895
calledby hashable, 1060
calledby mkEvalable, 903
calledby spadClosure?, 607
calledby spadTrace,isTraceable, 866
calledby spadTrace, 867
calledby spadUntrace, 886
calledby tracelet, 894
bpitrace
calledby spadTrace, 867
bpiuntrace
calledby spadUntrace, 886
break, 896
calledby letPrint2, 877
calledby letPrint3, 878
calledby letPrint, 875
calls MONITOR,EVALTRAN, 896
calls enable-backtrace, 896
calls interrupt, 896
calls sayBrightly, 896
uses /breakcondition, 896
defun, 896
breaklet, 895
calls assoc, 895
calls bpiname, 895
calls compileBoot, 895
calls delete, 895
calls gensymp, 895
calls lassoc, 895
calls setletprintflag, 895
calls stupidIsSpadFunction, 895
calls union, 895
uses $QuickLet, 895
uses $letAssoc, 895
uses $traceletFunctions, 895
defun, 895
bright
calledby ?t, 893
calledby commandAmbiguityError, 448
calledby displayCondition, 461
calledby displayFrameNames, 559
calledby displayMacro, 450
calledby displayModemap, 462
calledby displayMode, 462
calledby displayProperties,sayFunctionDeps, 454
calledby displayProperties, 458
calledby displaySetOptionInformation, 647
calledby displaySetVariableSettings, 649
calledby letPrint2, 877
calledby letPrint3, 878
calledby letPrint, 875
calledby pcounters, 850
calledby pspacers, 849
calledby ptimers, 849
calledby reportOperations, 807
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calledby set1, 800
calledby setFortDir, 718
calledby setFortPers, 748
calledby setFortTmpDir, 715
calledby setOutputCharacters, 758
calledby setStreamsCalculate, 793
calledby spadUntrace, 886
calledby zsystemdevelopment1, 944
brightprint, 1041
calledby saybrightly1, 1043
calls messageprint, 1041
defun, 1041
brightprint-0, 1042
calledby saybrightly1, 1043
calls messageprint-1, 1042
defun, 1042
browse
calls AXSERV;axServer;IMV;2, 489
calls AxiomServer, 489
calls loadLib, 489
calls set, 489
browse help page, 485
manpage, 485
browseOpen, 996
calls unsqueeze, 996
uses *allconstructors*, 996
uses *browse-stream*, 996
uses *browse-stream-stamp*, 996
uses $spadroot, 996
defun, 996
browseopen
calledby resethashtables, 991
calledby restart0, 19
browserAutoloadOnceTrigger

calledby make-databases, 1010
buildLibdb
calledby make-databases, 1010
bumperrorcount
calledby spad-syntax-error, 959
bvec-and, 1058
defun, 1058
bvec-concat, 1057
defun, 1057
bvec-copy, 1057
defun, 1057
bvec-elt, 1056
defmacro, 1056
bvec-equal, 1057
defun, 1057
bvec-greater, 1058
defun, 1058
bvec-make-full, 1056
defun, 1056
bvec-nand, 1059
defun, 1059
bvec-nor, 1059
defun, 1059
bvec-not, 1059
defun, 1059
bvec-or, 1058
defun, 1058
bvec-setelt, 1057
defmacro, 1057
bvec-size, 1057
defmacro, 1057
bvec-xor, 1058
defun, 1058

cacheKeyedMsg, 327
calledby fetchKeyedMsg, 326
uses *msghash*, 327
catches, 327
defun, 327
throws, 327
CallerName
calledby processKeyedError, 351
canCoerce
calledby unifyStructVar, 427
canCoerceFrom
calledby hasCateSpecial, 432
Catch
calledby intloopSpadProcess, 65
CatchAsCan
calledby intloopSpadProcess, 65
catches
cacheKeyedMsg, 327
executeQuietCommand, 47
InterpExecuteSpadSystemCommand, 32
interpretTopLevel, 53
intloop, 26
intloopSpadProcess, 64, 65
letPrint2, 877
letPrint3, 878
monitor-file, 1153
monitor-readinterp, 1160
monitor-spadfile, 1162
npListAndRecover, 187
npParse, 139
runspad, 21
safeWritify, 602
ScanOrPairVec, 612
serverReadLine, 44
zsystemdevelopment1, 944
categoryForm?
calledby evaluateType1, 907
calledby loadLib, 1029
calledby localnrlib, 1007
calledby make-databases, 1010
categoryOpen, 997
calls unsqueeze, 997
uses *category-stream*, 997
uses *category-stream-stamp*, 997
uses *hasCategory-hash*, 997
uses $spadroot, 997
defun, 997
categoryopen
calledby resethashtables, 991
calledby restart0, 19
cd help page, 491
manpage, 491
cdancols, 1072
defmacro, 1072
cdanrows, 1071
defmacro, 1071
cdaref2, 1070
defmacro, 1070

cdelt, 1067
  defmacro, 1067
cdlen, 1068
  defmacro, 1068
cdsetaref2, 1071
  defmacro, 1071
cdsetelt, 1068
  defmacro, 1068
centerAndHighlight
  calledby displayExposedConstructors, 696
  calledby displayExposedGroups, 696
  calledby displayHiddenConstructors, 697
  calledby displayOperationsFromLisplib, 812
  calledby displaySetOptionInformation, 647
  calledby displaySetVariableSettings, 649
  calledby filterAndFormatConstructors, 934
  calledby printSynonyms, 470
  calledby reportOpsFromLisplib, 810
  calledby reportOpsFromUnitDirectly, 814
  calledby setExposeAddConstr, 692
  calledby setExposeAddGroup, 691
  calledby setExposeAdd, 689
  calledby setExposeDropConstr, 695
  calledby setExposeDropGroup, 694
  calledby setExposeDrop, 693
  calledby trace1, 838
  calledby whatCommands, 931
  calledby workfilesSpad2Cmd, 939
changeHistListLen, 585
  calledby historySpad2Cmd, 578
  calls sayKeyedMsg, 585
  calls spaddifference, 585
  uses $HistListAct, 585
  uses $HistListLen, 585
  uses $HistList, 585
  defun, 585
changeToNamedInterpreterFrame, 556
  calledby serverReadLine, 44
  calls findFrameInRing, 556
  calls nremove, 556
  calls updateCurrentInterpreterFrame, 556
  calls updateFromCurrentInterpreterFrame, 556
  uses $interpreterFrameRing, 556
  defun, 556
char
  calledby incCommand?, 100
  calledby makeMsgFromLine, 369
charDigitVal, 613
  calledby gensymInt, 612
  calls error, 613
  calls spaddifference, 613
  defun, 613
charPosition
  calledby alqlGetOrigin, 1073
  calledby alqlGetParams, 1074
  calledby removeUndoLines, 923
cleanline, 525
  calledby describeSpad2Cmd, 525
  defun, 525
cleanupLine, 536
  defun, 536
cleanupline
  calledby sayexample, 536
clear, 495
  calls clearSpad2Cmd, 495
  defun, 495
clear help page, 493
  manpage, 493
clearAllSlams
  calledby updateDatabase, 1009
clearClams
  calledby clearCmdCompletely, 498
  calledby setExposeAddConstr, 692
  calledby setExposeAddGroup, 690
  calledby setExposeDropConstr, 695
  calledby setExposeDropGroup, 694
  calledby updateDatabase, 1009
clearCmdAll, 499
  calledby clearCmdCompletely, 498
  calledby clearFrame, 919
  calledby clearSpad2Cmd, 496
  calls clearCmdSortedCaches, 499
  calls clearMacroTable, 499
  calls deleteFile, 499
  calls histFileName, 499
  calls resetInCoreHist, 499
  calls sayKeyedMsg, 499
  calls untraceMapSubNames, 499
  calls updateCurrentInterpreterFrame, 499
  uses $InteractiveFrame, 499

uses $currentLine, 500
uses $frameMessages, 500
uses $frameRecord, 499
uses $internalHistoryTable, 499
uses $interpreterFrameName, 500
uses $previousBindings, 499
uses $useInternalHistoryTable, 499
uses $variableNumberAlist, 499
defun, 499
clearCmdCompletely, 498
calledby clearSpad2Cmd, 496
calls clearClams, 498
calls clearCmdAll, 498
calls clearConstructorCaches, 498
calls reclaim, 498
calls sayKeyedMsg, 498
uses $CatOfCatDatabase, 498
uses $DomOfCatDatabase, 498
uses $JoinOfCatDatabase, 498
uses $JoinOfDomDatabase, 498
uses $attributeDb, 498
uses $existingFiles, 499
uses $functionTable, 499
uses $localExposureDataDefault, 499
uses $localExposureData, 498
uses $xdatabase, 498
defun, 498
clearCmdExcept, 500
calledby clearSpad2Cmd, 496
calls clearCmdParts, 501
calls object2String, 501
calls stringPrefix?, 500
uses $clearOptions, 501
defun, 500
clearCmdParts, 501
calledby clearCmdExcept, 501
calledby clearSpad2Cmd, 496
calledby importFromFrame, 559
calls assocleft, 501
calls assoc, 501
calls boot-equal, 501
calls clearDependencies, 501
calls clearParserMacro, 501
calls deleteAssoc, 501
calls exit, 501
calls fixObjectForPrinting, 501
calls getInterpMacroNames, 501
calls getParserMacroNames, 501
calls get, 501
calls isMap, 501
calls member, 501
calls modes, 501
calls pname, 501
calls recordNewValue, 501
calls recordOldValue, 501
calls remdup, 501
calls sayKeyedMsg, 501
calls sayMessage, 501
calls selectOptionLC, 501
calls seq, 501
calls types, 501
calls untraceMapSubNames, 501
calls values, 501
uses $InteractiveFrame, 501
uses $clearOptions, 501
uses $e, 501
defun, 501
clearCmdSortedCaches, 497
calledby clearCmdAll, 499
calledby clearSpad2Cmd, 496
calledby restoreHistory, 593
calls compiledLookupCheck, 497
calls spadcall, 497
uses $ConstructorCache, 497
uses $Void, 497
uses $lookupDefaults, 497
defun, 497
clearConstructorCache
calledby loadLibNoUpdate, 1031
calledby loadLib, 1029
clearConstructorCaches
calledby clearCmdCompletely, 498
clearDependencies
calledby clearCmdParts, 501
clearFrame, 919
calls clearCmdAll, 919
uses $frameRecord, 919
uses $previousBindings, 919
defun, 919
clearMacroTable, 500
calledby clearCmdAll, 499
uses $pfMacros, 500

defun, 500
clearParserMacro, 449
calledby clearCmdParts, 501
calls assoc, 449
calls ifcdr, 449
calls remalist, 449
uses $pfMacros, 449
defun, 449
clearSpad2Cmd, 496
calledby clear, 495
calledby historySpad2Cmd, 578
calledby restoreHistory, 593
calls clearCmdAll, 496
calls clearCmdCompletely, 496
calls clearCmdExcept, 496
calls clearCmdParts, 496
calls clearCmdSortedCaches, 496
calls sayKeyedMsg, 496
calls selectOptionLC, 496
calls updateCurrentInterpreterFrame, 496
uses $clearExcept, 496
uses $clearOptions, 496
uses $options, 496
defun, 496
clef, 386
syntax, 386
close, 506
calls closeInterpreterFrame, 506
calls queryClients, 507
calls queryUserKeyedMsg, 506
calls selectOptionLC, 506
calls sockSendInt, 506
calls string2id-n, 507
calls throwKeyedMsg, 506
calls upcase, 506
uses $CloseClient, 507
uses $SessionManager, 507
uses $SpadServer, 507
uses $currentFrameNum, 507
uses $options, 507
defun, 506
close help page, 505
manpage, 505
closeInterpreterFrame, 558
calledby close, 506
calledby frameSpad2Cmd, 562
calls $erase, 558
calls framename, 558
calls makeHistFileName, 558
calls throwKeyedMsg, 558
calls updateFromCurrentInterpreterFrame, 558
uses $interpreterFrameName, 558
uses $interpreterFrameRing, 558
defun, 558
clrhash
calledby processInteractive, 50
cmpnote, 28
defun, 28
cnstructSubst
calledby hasSig, 421
coerceInteractive
calledby coerceSpadArgs2E, 855
calledby coerceSpadFunValue2E, 858
calledby coerceTraceArgs2E, 854
calledby coerceTraceFunValue2E, 857
calledby interpret2, 55
coerceOrRetract
calledby evaluateType1, 908
coerceSpadArgs2E, 855
calledby coerceTraceArgs2E, 854
calls coerceInteractive, 855
calls exit, 855
calls objNewWrap, 855
calls objValUnwrap, 855
calls seq, 855
uses $OutputForm, 855
uses $streamCount, 855
uses $tracedSpadModemap, 855
defun, 855
coerceSpadFunValue2E, 858
calledby coerceTraceFunValue2E, 857
calls coerceInteractive, 858
calls objNewWrap, 858
calls objValUnwrap, 858
uses $OutputForm, 858
uses $streamCount, 858
uses $tracedSpadModemap, 858
defun, 858
coerceTraceArgs2E, 854
calls coerceInteractive, 854
calls coerceSpadArgs2E, 854

calls objNewWrap, 854
calls objValUnwrap, 854
calls pname, 854
calls spadsysnamep, 854
uses $OutputForm, 854
uses $mathTraceList, 854
uses $tracedMapSignatures, 854
defun, 854
coerceTraceFunValue2E, 857
calls coerceInteractive, 857
calls coerceSpadFunValue2E, 857
calls lassoc, 857
calls objNewWrap, 857
calls objValUnwrap, 857
calls pname, 857
calls spadsysnamep, 857
uses $OutputForm, 857
uses $mathTraceList, 857
uses $tracedMapSignatures, 857
defun, 857
collection, 387
syntax, 387
commandAmbiguityError, 448
calledby commandErrorIfAmbiguous, 466
calledby commandErrorMessage, 445
calledby traceOptionError, 847
calledby userLevelErrorMessage, 446
calls bright, 448
calls sayKeyedMsg, 448
calls sayMSG, 448
calls terminateSystemCommand, 448
defun, 448
commandError, 445
calls commandErrorMessage, 445
defun, 445
commandErrorIfAmbiguous, 466
calls commandAmbiguityError, 466
uses $oldline, 466
uses line, 466
defun, 466
commandErrorMessage, 445
calledby commandError, 445
calledby optionError, 445
calls commandAmbiguityError, 445
calls sayKeyedMsg, 446
calls terminateSystemCommand, 446
uses $oldline, 446
uses line, 446
defun, 445
commandsForUserLevel, 444
calledby synonymsForUserLevel, 825
calledby systemCommand, 444
calledby unAbbreviateKeyword, 465
calledby whatCommands, 932
calls satisfiesUserLevel, 444
defun, 444
commandUserLevelError, 446
calls userLevelErrorMessage, 446
defun, 446
compareposns, 368
calledby erMsgCompare, 368
calls poCharPosn, 368
calls poGlobalLinePosn, 368
defun, 368
compFailure
calledby getAndEvalConstructorArgument, 948
compile help page, 509
manpage, 509
compileBoot, 897
calledby breaklet, 895
calledby tracelet, 894
calls /D,1, 897
defun, 897
compiledLookup, 1061
calledby compiledLookupCheck, 497
calledby hashable, 1060
calls NRTevalDomain, 1061
calls isDomain, 1061
defun, 1061
compiledLookupCheck, 497
calledby clearCmdSortedCaches, 497
calls compiledLookup, 497
calls formatSignature, 498
calls keyedSystemError, 497
defun, 497
compressOpen, 1015
calls DaaseName, 1015
uses *compress-stream*, 1015
uses *compress-stream-stamp*, 1015
uses *compressVectorLength*, 1015
uses *compressvector*, 1015

uses $spadroot, 1015
defun, 1015
compressopen
calledby resethashtables, 991
calledby restart0, 18
CONCAT
calledby xlSkip, 89
concat, 1041
calledby copyright, 518
calledby dewritify,dewritifyInner, 608
calledby displayCondition, 461
calledby displayModemap, 462
calledby displayMode, 462
calledby displaySetOptionInformation, 647
calledby displaySetVariableSettings, 649
calledby displayType, 456
calledby displayValue, 455
calledby doSystemCommand, 442
calledby evalDomain, 903
calledby getTraceOption, 844
calledby handleNoParseCommands, 466
calledby incLude1, 81
calledby inclmsgIfSyntax, 97
calledby intloopReadConsole, 30
calledby lffloat, 125
calledby lfrinteger, 131
calledby mkprompt, 42
calledby ncloopIncFileName, 618
calledby newHelpSpad2Cmd, 568
calledby pcounters, 850
calledby printLabelledList, 470
calledby processSynonyms, 33
calledby pspacers, 849
calledby ptimers, 849
calledby removeUndoLines, 924
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calledby reportUndo, 918
calledby resetCounters, 848
calledby resetSpacers, 848
calledby resetTimers, 848
calledby scanExponent, 124
calledby scanNumber, 129
calledby scanS, 128
calledby scanW, 125
calledby setOutputAlgebra, 754
calledby setOutputCharacters, 758
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby spleI1, 120
calledby summary, 822
calledby traceDomainConstructor, 871
calledby traceReply, 889
calledby undoCount, 919
calledby untraceDomainConstructor, 873
calledby writeInputLines, 583
calls string-concatenate, 1041
defun, 1041
consoleinputp
calledby spad-syntax-error, 959
constoken, 111
calledby lineoftoks, 109
calledby scanToken, 112
calls ncPutQ, 111
defun, 111
constructor?
calledby addTraceItem, 892
calledby evaluateType1, 907
calledby evaluateType, 906
calledby hasSig, 421
calledby mkEvalable, 903
calledby reportOpsFromLisplib, 810
calledby transTraceItem, 853
calledby unifyStructVar, 427
calledby updateDatabase, 1009
calledby writify,writifyInner, 603
constructSubst, 432
calledby hasAtt, 422
calledby hasCaty, 418
calledby spadTrace, 867
calls internl, 432
calls stringimage, 432
defun, 432
contained
calledby isPartialMode, 418
calledby unifyStructVar, 427
containsVariables
calledby hasCate, 431

containsVars, 428
  calledby hasCateSpecial, 432
  calledby unifyStructVar, 427
  calls containsVars1, 429
  calls isPatternVar, 428
  defun, 428
containsVars1, 429
  calledby containsVars1, 429
  calledby containsVars, 429
  calls containsVars1, 429
  calls isPatternVar, 429
  defun, 429
copy
  calledby makeInitialModemapFrame, 37
  calledby resetWorkspaceVariables, 646
  calledby undoSteps, 920
  calledby untrace, 852
copyright, 518
  calls concat, 518
  calls getenviron, 518
  calls obey, 518
  defun, 518
copyright help page, 513
  manpage, 513
cot, 1089
  defun, 1089
coth, 1091
  defun, 1091
createCurrentInterpreterFrame, 553
  calledby updateCurrentInterpreterFrame, 555
  uses $HiFiAccess, 553
  uses $HistListAct, 553
  uses $HistListLen, 553
  uses $HistList, 553
  uses $HistRecord, 553
  uses $IOindex, 553
  uses $InteractiveFrame, 553
  uses $internalHistoryTable, 553
  uses $interpreterFrameName, 553
  uses $localExposureData, 553
  defun, 553
credits, 1, 521
  usedby credits, 521
  uses credits, 521
  defun, 521
  defvar, 1
credits help page, 521
  manpage, 521
csc, 1090
  defun, 1090
csch, 1091
  defun, 1091
curinstream, 23
  usedby ncIntLoop, 25
  defvar, 23
curoutstream, 23
  usedby ncIntLoop, 26
  defvar, 23
current-fragment
  usedby ioclear, 962
current-line
  usedby ioclear, 962
current-token
  usedby token-stack-show, 961
currenttime
  calledby mkprompt, 42

DaaseName, 1014
  calledby compressOpen, 1015
  calledby interpOpen, 995
  calls getEnv, 1014
  uses $spadroot, 1014
  defun, 1014
dancols, 1070
  defmacro, 1070
danrows, 1070
  defmacro, 1070
daref2, 1069
  defmacro, 1069
database, 986
  defstruct, 986
dbPart
  calledby alqlGetKindString, 1074
  calledby alqlGetOrigin, 1073
  calledby alqlGetParams, 1074
dbSplitLibdb
  calledby make-databases, 1010
dc1
  calledby reportOpsFromLisplib, 810
debugmode
  usedby spad-syntax-error, 959

decideHowMuch, 357
calledby getPosStL, 354
calls poFileName, 357
calls poLinePosn, 357
calls poNopos?, 357
calls poPosImmediate?, 357
defun, 357
defaultTargetFE, 435
calledby defaultTargetFE, 435
calledby hasCateSpecialNew, 433
calls defaultTargetFE, 435
calls ifcar, 435
calls isEqualOrSubDomain, 435
calls typeIsASmallInteger, 435
local ref $FunctionalExpression, 435
local ref $Integer, 435
local ref $RationalNumber, 435
local ref $Symbol, 435
defun, 435
defiostream, 972
calledby reportOpsFromLisplib1, 809
calledby reportOpsFromUnitDirectly1, 817
calledby sayMSG2File, 329
calledby setOutputAlgebra, 754
calledby setOutputFormula, 781
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby writeInputLines, 583
calledby zsystemdevelopment1, 944
defun, 972
defmacro
assq, 1044
bit-to-truth, 1056
bvec-elt, 1056
bvec-setelt, 1057
bvec-size, 1057
cdancols, 1072
cdanrows, 1071
cdaref2, 1070
cdelt, 1067
cdlen, 1068
cdsetaref2, 1071
cdsetelt, 1068
dancols, 1070
danrows, 1070
daref2, 1069
delt, 1067
DFAcos, 1084
DFAcosh, 1087
DFAdd, 1080
DFAsin, 1084
DFAsinh, 1086
DFAtan, 1085
DFAtan2, 1085
DFAtanh, 1087
DFCos, 1084
DFCosh, 1086
DFDivide, 1081
DFEql, 1081
DFExp, 1083
DFExpt, 1083
DFIntegerDivide, 1082
DFIntegerExpt, 1083
DFIntegerMultiply, 1080
DFLessThan, 1079
DFLog, 1082
DFLogE, 1082
DFMax, 1081
DFMin, 1081
DFMinusp, 1079
DFMultiply, 1080
DFSin, 1083
DFSinh, 1085
DFSqrt, 1082
DFSubtract, 1080
DFTan, 1084
DFTanh, 1086
DFUnaryMinus, 1079
DFZerop, 1079
dlen, 1066
dsetaref2, 1069
dsetelt, 1067
elt32, 1048
funfind, 864
hget, 1038
idChar?, 125
identp, 1040
make-cdouble-matrix, 1070
make-cdouble-vector, 1067

make-double-matrix, 1069
make-double-matrix1, 1069
make-double-vector, 1066
make-double-vector1, 1066
qsabsval, 1054
qsadd1, 1053
qsdifference, 1052
qslessp, 1053
qsmax, 1055
qsmin, 1055
qsminus, 1053
qsoddp, 1054
qsplus, 1054
qssub1, 1053
qstimes, 1054
qszerop, 1055
qv32len, 1048
Rest, 77
setelt32, 1048
spadConstant, 1139
startsId?, 1038
trapNumericErrors, 1065
truth-to-bit, 1056
while, 1033
whileWithResult, 1034
defstruct
database, 986
libstream, 1148
monitor-data, 1148
defun
/read, 640
/tracereply, 884
?t, 892
abbQuery, 532
abbreviations, 479
abbreviationsSpad2Cmd, 479
acot, 1089
acoth, 1092
acsc, 1090
acsch, 1091
addBinding, 953
addBindingInteractive, 957
addInputLibrary, 660
addNewInterpreterFrame, 557
addoperations, 998
addTraceItem, 892
algCoerceInteractive, 1051
allConstructors, 1027
allOperations, 1027
alqlGetKindString, 1074
alqlGetOrigin, 1073
alqlGetParams, 1074
alreadyOpened?, 361
apropos, 935
asec, 1090
asech, 1092
assertCond, 96
augmentTraceNames, 862
axiomVersion, 472
basicLookup, 1061
basicLookupCheckDefaults, 1063
BooleanEquality, 1055
break, 896
breaklet, 895
brightprint, 1041
brightprint-0, 1042
browseOpen, 996
bvec-and, 1058
bvec-concat, 1057
bvec-copy, 1057
bvec-equal, 1057
bvec-greater, 1058
bvec-make-full, 1056
bvec-nand, 1059
bvec-nor, 1059
bvec-not, 1059
bvec-or, 1058
bvec-xor, 1058
cacheKeyedMsg, 327
categoryOpen, 997
changeHistListLen, 585
changeToNamedInterpreterFrame, 556
charDigitVal, 613
cleanline, 525
cleanupLine, 536
clear, 495
clearCmdAll, 499
clearCmdCompletely, 498
clearCmdExcept, 500
clearCmdParts, 501
clearCmdSortedCaches, 497
clearFrame, 919

clearMacroTable, 500
clearParserMacro, 449
clearSpad2Cmd, 496
close, 506
closeInterpreterFrame, 558
cmpnote, 28
coerceSpadArgs2E, 855
coerceSpadFunValue2E, 858
coerceTraceArgs2E, 854
coerceTraceFunValue2E, 857
commandAmbiguityError, 448
commandError, 445
commandErrorIfAmbiguous, 466
commandErrorMessage, 445
commandsForUserLevel, 444
commandUserLevelError, 446
compareposns, 368
compileBoot, 897
compiledLookup, 1061
compiledLookupCheck, 497
compressOpen, 1015
concat, 1041
constoken, 111
constructSubst, 432
containsVars, 428
containsVars1, 429
copyright, 518
cot, 1089
coth, 1091
createCurrentInterpreterFrame, 553
credits, 521
csc, 1090
csch, 1091
DaaseName, 1014
decideHowMuch, 357
defaultTargetFE, 435
defiostream, 972
Delay, 103
deldatabase, 1001
deleteFile, 1036
describe, 524
describeFortPersistence, 748
describeInputLibraryArgs, 660
describeOutputLibraryArgs, 657
describeSetFortDir, 718
describeSetFortTmpDir, 716
describeSetLinkerArgs, 720
describeSetNagHost, 746
describeSetOutputAlgebra, 756
describeSetOutputFormula, 784
describeSetOutputFortran, 763
describeSetOutputHtml, 774
describeSetOutputMathml, 770
describeSetOutputOpenMath, 779
describeSetOutputTex, 790
describeSetStreamsCalculate, 793
describeSpad2Cmd, 524
desiredMsg, 350
dewritify, 611
dewritify,dewritifyInner, 608
diffAlist, 914
digit?, 119
digitp, 1039
DirToString, 1076
disableHist, 599
display, 531
displayCondition, 461
displayExposedConstructors, 696
displayExposedGroups, 696
displayFrameNames, 559
displayHiddenConstructors, 697
displayMacro, 449
displayMacros, 534
displayMode, 462
displayModemap, 462
displayOperations, 533
displayOperationsFromLisplib, 812
displayParserMacro, 460
displayProperties, 457
displayProperties,sayFunctionDeps, 452
displaySetOptionInformation, 647
displaySetVariableSettings, 649
displayType, 456
displayValue, 455
displayWorkspaceNames, 450
divide2, 1072
domainToGenvar, 851
domArg, 420
domArg2, 420
doSystemCommand, 442
dqAppend, 342
dqConcat, 341

dqToList, 342
dqUnit, 341
dropInputLibrary, 661
dumbTokenize, 463
edit, 540
editFile, 541
editSpad2Cmd, 540
emptyInterpreterFrame, 552
enPile, 338
eofp, 973
eqpileTree, 337
erMsgCompare, 368
erMsgSep, 368
erMsgSort, 367
evalCategory, 949
evalDomain, 903
evaluateSignature, 910
evaluateType, 906
evaluateType1, 907
ExecuteInterpSystemCommand, 33
executeQuietCommand, 47
fetchKeyedMsg, 326
fetchOutput, 596
fillerSpaces, 20
filterAndFormatConstructors, 934
filterListOfStrings, 932
filterListOfStringsWithFn, 933
fin, 544
findFrameInRing, 555
firstTokPosn, 339
fixObjectForPrinting, 452
flatten, 527
flattenOperationAlist, 873
float2Sex, 303
fnameDirectory, 1076
fnameExists?, 1077
fnameMake, 1075
fnameName, 1077
fnameNew, 1078
fnameReadable?, 1077
fnameType, 1077
fnameWritable?, 1078
frame, 561
frameEnvironment, 552
frameName, 548
frameNames, 552
frameSpad2Cmd, 562
From, 378
FromTo, 378
functionp, 1041
funfind,LAM, 864
genDomainTraceName, 852
gensymInt, 612
get-current-directory, 36
getAliasIfTracedMapParameter, 879
getAndEvalConstructorArgument, 948
getAndSay, 457
getBpiNameIfTracedMap, 880
getBrowseDatabase, 1074
getdatabase, 1001
getDirectoryList, 974
getenviron, 31
getFirstWord, 465
getKeyedMsg, 327
getLinePos, 370
getLineText, 370
getMapSig, 843
getMapSubNames, 859
getMsgArgL, 347
getMsgCatAttr, 356
getMsgFTTag?, 357
getMsgInfoFromKey, 364
getMsgKey, 346
getMsgKey?, 360
getMsgLitSym, 360
getMsgPos, 357
getMsgPos2, 376
getMsgPosTagOb, 346
getMsgPrefix, 347
getMsgPrefix?, 348
getMsgTag, 348
getMsgTag?, 348
getMsgText, 347
getMsgToWhere, 361
getOplistForConstructorForm, 816
getOplistWithUniqueSignatures, 817
getOption, 882
getParserMacroNames, 449
getPosStL, 354
getPreStL, 353
getPreviousMapSubNames, 860
getProplist, 954

getrefv32, 1048
getStFromMsg, 352
getSystemCommandLine, 825
getTraceOption, 844
getTraceOption,hn, 843
getTraceOptions, 842
getWorkspaceNames, 451
handleNoParseCommands, 441
handleParsedSystemCommands, 464
handleTokensizeSystemCommands, 443
hasAtt, 422
hasAttSig, 424
hasCate, 431
hasCate1, 425
hasCateSpecial, 432
hasCateSpecialNew, 433
hasCatExpression, 425
hasCaty, 418
hasCaty1, 429
hashable, 1060
hasOptArgs?, 307
hasOption, 447
hasPair, 881
hasSig, 421
hasSigAnd, 423
hasSigOr, 424
help, 568
helpSpad2Cmd, 568
histFileErase, 613
histFileName, 576
histInputFileName, 576
history, 578
historySpad2Cmd, 578
hkeys, 1038
hput, 1038
ifCond, 89
importFromFrame, 559
incActive?, 103
incAppend, 87
incAppend1, 87
incBiteOff, 619
incClassify, 99
incCommand?, 100
incCommandTail, 101
incConsoleInput, 102
incDrop, 101
incFileInput, 102
incFileName, 618
incHandleMessage, 76
incIgen, 75
incIgen1, 75
inclFname, 102
incLine, 87
incLine1, 88
inclmsgCannotRead, 92
inclmsgCmdBug, 98
inclmsgConActive, 93
inclmsgConsole, 94
inclmsgConStill, 94
inclmsgFileCycle, 92
inclmsgFinSkipped, 95
inclmsgIfBug, 97
inclmsgIfSyntax, 97
inclmsgNoSuchFile, 91
inclmsgPrematureEOF, 88
inclmsgPrematureFin, 95
inclmsgSay, 90
incLude, 76
incLude1, 81
incNConsoles, 103
incPrefix?, 100
incRenumber, 74
incRenumberItem, 76
incRenumberLine, 75
incRgen, 103
incRgen1, 104
incStream, 73
incString, 39
incZip, 74
incZip1, 74
init-boot/spad-reader, 958
init-memory-config, 34
initHist, 577
initHistList, 577
initial-getdatabase, 992
initializeInterpreterFrameRing, 551
initializeSetVariables, 645
initImPr, 365
initroot, 35
initToWhere, 366
insertpile, 333
insertPos, 377

integer-decode-float-denominator, 1088
integer-decode-float-exponent, 1088
integer-decode-float-numerator, 1087
integer-decode-float-sign, 1088
InterpExecuteSpadSystemCommand, 32
interpFunctionDepAlists, 461
interpOpen, 994
interpret, 54
interpret1, 54
interpret2, 55
interpretTopLevel, 53
intInterpretPform, 67
intloop, 26
intloopEchoParse, 69
intloopInclude, 63
intloopInclude0, 63
intloopPrefix?, 36
intloopProcess, 64
intloopProcessString, 37
intloopReadConsole, 30
intloopSpadProcess, 64
intloopSpadProcess,interp, 65
intnplisp, 36
intProcessSynonyms, 33
ioclear, 962
iostat, 960
isDomainOrPackage, 865
isDomainValuedVariable, 949
isEqualOrSubDomain, 436
isExposedConstructor, 812
isgenvar, 876
isIntegerString, 464
isInterpOnlyMap, 862
isListOfIdentifiers, 858
isListOfIdentifiersOrStrings, 859
isPartialMode, 418
isSharpVar, 876
isSharpVarWithNum, 876
isSubForRedundantMapName, 863
isSystemDirectory, 1030
isTraceGensym, 865
isUncompiledMap, 861
justifyMyType, 62
keyword, 118
keyword?, 118
lassocSub, 861
lastTokPosn, 339
leader?, 349
leaveScratchpad, 635
letPrint, 875
letPrint2, 877
letPrint3, 878
lfcomment, 114
lferror, 133
lffloat, 125
lfid, 112
lfinteger, 131
lfkey, 119
lfnegcomment, 115
lfrinteger, 131
lfspaces, 127
lfstring, 128
library, 1005
line?, 349
lineoftoks, 108
listConstructorAbbreviations, 480
listDecideHowMuch, 359
listOutputter, 352
lnCreate, 343
lnExtraBlanks, 343
lnFileName, 345
lnFileName?, 345
lnGlobalNum, 344
lnImmediate?, 345
lnLocalNum, 344
lnPlaceOfOrigin, 344
lnSetGlobalNum, 344
lnString, 343
load, 625
loadFunctor, 1032
loadLib, 1029
loadLibNoUpdate, 1031
localdatabase, 1005
localnrlib, 1007
lookupInDomainVector, 1063
loopIters2Sex, 309
lotsof, 1037
ltrace, 628
mac0Define, 228
mac0ExpandBody, 222
mac0Get, 226
mac0GetName, 224

mac0InfiniteExpansion, 223
mac0InfiniteExpansion,name, 224
mac0MLambdaApply, 221
mac0SubstituteOuter, 229
macApplication, 221
macExpand, 220
macId, 225
macLambda, 226
macLambda,mac, 227
macLambdaParameterHandling, 229
macMacro, 227
macroExpanded, 220
macSubstituteId, 230
macSubstituteOuter, 228
macWhere, 226
macWhere,mac, 226
make-absolute-filename, 36
make-appendstream, 972
make-databases, 1009
make-instream, 971
make-outstream, 971
makeByteWordVec2, 1139
makeFullNamestring, 975
makeHistFileName, 575
makeInitialModemapFrame, 37
makeInputFilename, 973
makeLeaderMsg, 375
makeMsgFromLine, 369
makeOrdinal, 910
makePathname, 1036
makeStream, 973
manexp, 1088
mapLetPrint, 874
member, 1042
mergePathnames, 1035
messageprint, 1042
messageprint-1, 1043
messageprint-2, 1043
mkDomPvar, 430
mkEvalable, 903
mkEvalableMapping, 905
mkEvalableRecord, 905
mkEvalableUnion, 905
mkLineList, 70
mkprompt, 42
monitor-add, 1150
monitor-apropos, 1163
monitor-autoload, 1158
monitor-checkpoint, 1155
monitor-decr, 1152
monitor-delete, 1150
monitor-dirname, 1158
monitor-disable, 1151
monitor-enable, 1150
monitor-end, 1149
monitor-exposedp, 1160
monitor-file, 1153
monitor-help, 1156
monitor-incr, 1152
monitor-info, 1153
monitor-inittable, 1148
monitor-libname, 1159
monitor-nrlib, 1159
monitor-parse, 1162
monitor-percent, 1163
monitor-readinterp, 1160
monitor-report, 1161
monitor-reset, 1151
monitor-restore, 1155
monitor-results, 1149
monitor-spadfile, 1162
monitor-tested, 1154
monitor-untested, 1153
monitor-write, 1154
msgCreate, 345
msgImPr?, 356
msgNoRep?, 373
msgOutputter, 351
msgText, 61
myWritable?, 1078
namestring, 1034
ncAlist, 413
ncBug, 366
ncConversationPhase, 68
ncConversationPhase,wrapup, 68
ncEltQ, 414
ncError, 69
ncHardError, 350
ncIntLoop, 25
ncloopCommand, 474
ncloopDQlines, 71
ncloopEscaped, 37

ncloopIncFileName, 618
ncloopInclude, 618
ncloopInclude0, 73
ncloopInclude1, 617
ncloopParse, 38
ncloopPrefix?, 475
ncloopPrintLines, 70
ncParseFromString, 1052
ncPutQ, 414
ncSoftError, 349
ncTag, 413
ncTopLevel, 25
newHelpSpad2Cmd, 568
next, 38
next-lines-clear, 962
next-lines-show, 961
next1, 38
nextInterpreterFrame, 556
nextline, 110
nonBlank, 71
npADD, 156
npAdd, 157
npAmpersand, 202
npAmpersandFrom, 200
npAndOr, 179
npAngleBared, 184
npAnyNo, 160
npApplication, 160
npApplication2, 161
npArith, 198
npAssign, 214
npAssignment, 215
npAssignVariable, 215
npAtom1, 181
npAtom2, 157
npBacksetElse, 195
npBackTrack, 146
npBDefinition, 183
npboot, 468
npBPileDefinition, 186
npBraced, 184
npBracked, 184
npBracketed, 183
npBreak, 172
npBy, 197
npCategory, 151
npCategoryL, 150
npCoerceTo, 218
npColon, 215
npColonQuery, 217
npComma, 144
npCommaBackSet, 144
npCompMissing, 149
npConditional, 193
npConditionalStatement, 178
npConstTok, 182
npDDInfKey, 206
npDecl, 212
npDef, 185
npDefaultDecl, 168
npDefaultItem, 167
npDefaultItemlist, 166
npDefaultValue, 192
npDefinition, 165
npDefinitionItem, 165
npDefinitionlist, 191
npDefinitionOrStatement, 145
npDefn, 185
npDefTail, 192
npDiscrim, 195
npDisjand, 195
npDollar, 181
npDotted, 160
npElse, 194
npEncAp, 180
npEncl, 180
npEnclosed, 209
npEqKey, 143
npEqPeek, 150
npExit, 213
npExport, 169
npExpress, 177
npExpress1, 177
npFirstTok, 141
npFix, 164
npForIn, 175
npFree, 171
npFromdom, 200
npFromdom1, 201
npGives, 146
npId, 202
npImport, 178

npInfGeneric, 205
npInfixOp, 159
npInfixOperator, 158
npInfKey, 206
npInline, 172
npInterval, 197
npItem, 140
npItem1, 140
npIterate, 172
npIterator, 174
npIterators, 173
npLambda, 146
npLeftAssoc, 204
npLet, 164
npLetQualified, 164
nplisp, 468
npList, 153
npListAndRecover, 187
npListing, 153
npListofFun, 219
npLocal, 171
npLocalDecl, 170
npLocalItem, 170
npLocalItemlist, 169
npLogical, 195
npLoop, 173
npMacro, 162
npMatch, 148
npMDEF, 163
npMdef, 162
npMDEFinition, 163
npMissing, 149
npMissingMate, 213
npMoveTo, 189
npName, 202
npNext, 143
npNull, 331
npParened, 183
npParenthesize, 213
npParenthesized, 212
npParse, 139
npPDefinition, 181
npPileBracketed, 186
npPileDefinitionlist, 187
npPileExit, 214
npPop1, 142
npPop2, 142
npPop3, 142
npPower, 200
npPP, 207
npPPf, 209
npPPff, 208
npPPg, 208
npPrefixColon, 159
npPretend, 217
npPrimary, 155
npPrimary1, 162
npPrimary2, 156
npProcessSynonym, 469
npProduct, 200
npPush, 141
npPushId, 207
npQualDef, 143
npQualified, 145
npQualifiedDefinition, 145
npQualType, 179
npQualTypelist, 178
npQuiver, 196
npRecoverTrap, 188
npRelation, 196
npRemainder, 199
npRestore, 150
npRestrict, 218
npReturn, 176
npRightAssoc, 204
npRule, 191
npSCategory, 151
npSDefaultItem, 167
npSegment, 198
npSelector, 161
npSemiBackSet, 191
npSemiListing, 191
npSigDecl, 155
npSigItem, 154
npSigItemlist, 152
npSignature, 152
npSignatureDefinee, 154
npSingleRule, 192
npSLocalItem, 170
npSQualTypelist, 179
npState, 210
npStatement, 168

npSuch, 148
npSuchThat, 174
npSum, 199
npSymbolVariable, 203
npsynonym, 469
npSynthetic, 196
npsystem, 468
npTagged, 216
npTerm, 199
npTrap, 210
npTrapForm, 210
npTuple, 144
npType, 147
npTypedForm, 218
npTypedForm1, 216
npTypeStyle, 217
npTypeVariable, 154
npTypeVariablelist, 155
npTypified, 216
npTyping, 166
npVariable, 211
npVariablelist, 211
npVariableName, 211
npVoid, 177
npWConditional, 193
npWhile, 175
npWith, 148
npZeroOrMore, 175
NRTevalDomain, 1064
ofCategory, 417
oldCompLookup, 1064
oldHistFileName, 576
om-bindTCP, 1125
om-closeConn, 1124
om-closeDev, 1124
om-connectTCP, 1126
om-getApp, 1127
om-getAtp, 1128
om-getAttr, 1128
om-getBind, 1128
om-getBVar, 1128
om-getByteArray, 1129
om-getConnInDev, 1125
om-getConnOutDev, 1125
om-getEndApp, 1129
om-getEndAtp, 1129
om-getEndAttr, 1129
om-getEndBind, 1130
om-getEndBVar, 1130
om-getEndError, 1130
om-getEndObject, 1130
om-getError, 1131
om-getFloat, 1131
om-getInt, 1131
om-getObject, 1131
om-getString, 1132
om-getSymbol, 1132
om-getType, 1132
om-getVar, 1132
om-listCDs, 1122
om-listSymbols, 1122
om-makeConn, 1124
om-openFileDev, 1123
om-openStringDev, 1124
om-putApp, 1133
om-putAtp, 1133
om-putAttr, 1133
om-putBind, 1133
om-putBVar, 1134
om-putByteArray, 1134
om-putEndApp, 1134
om-putEndAtp, 1134
om-putEndAttr, 1135
om-putEndBind, 1135
om-putEndBVar, 1135
om-putEndError, 1135
om-putEndObject, 1136
om-putError, 1136
om-putFloat, 1136
om-putInt, 1136
om-putObject, 1137
om-putString, 1137
om-putSymbol, 1137
om-putVar, 1137
om-Read, 1121
om-setDevEncoding, 1123
om-stringPtrToString, 1138
om-stringToStringPtr, 1138
om-supportsCD, 1122
om-supportsSymbol, 1122
openOutputLibrary, 658
openserver, 977

operationOpen, 998
optionError, 445
optionUserLevelError, 446
opTran, 321
orderBySlotNumber, 883
parseAndInterpret, 48
parseFromString, 48
parseSystemCmd, 465
pathname, 1036
pathnameDirectory, 1035
pathnameName, 1034
pathnameType, 1034
pathnameTypeId, 1035
patternVarsOf, 319
patternVarsOf1, 319
pcounters, 850
pf0ApplicationArgs, 235
pf0AssignLhsItems, 254
pf0DefinitionLhsItems, 260
pf0FlattenSyntacticTuple, 235
pf0ForinLhs, 265
pf0FreeItems, 264
pf0LambdaArgs, 272
pf0LocalItems, 273
pf0LoopIterators, 274
pf0MLambdaArgs, 276
pf0SequenceArgs, 285
pf0TupleParts, 291
pf0WhereContext, 293
pf2Sex, 297
pf2Sex1, 298
pfAbSynOp, 410
pfAbSynOp?, 410
pfAdd, 250
pfAnd, 251
pfAnd?, 252
pfAndLeft, 252
pfAndRight, 253
pfAppend, 253
pfApplication, 251
pfApplication2Sex, 303
pfApplication?, 253
pfApplicationArg, 252
pfApplicationOp, 252
pfAssign, 253
pfAssign?, 254
pfAssignLhsItems, 254
pfAssignRhs, 254
pfAttribute, 251
pfBrace, 255
pfBraceBar, 255
pfBracket, 255
pfBracketBar, 255
pfBreak, 256
pfBreak?, 256
pfBreakFrom, 256
pfCharPosn, 233
pfCheckArg, 239
pfCheckId, 239
pfCheckItOut, 237
pfCheckMacroOut, 238
pfCoerceto, 257
pfCoerceto?, 257
pfCoercetoExpr, 257
pfCoercetoType, 257
pfCollect, 258
pfCollect1?, 240
pfCollect2Sex, 312
pfCollect?, 258
pfCollectArgTran, 315
pfCollectBody, 258
pfCollectIterators, 258
pfCollectVariable1, 240
pfCopyWithPos, 234
pfDefinition, 259
pfDefinition2Sex, 313
pfDefinition?, 259
pfDefinitionLhsItems, 259
pfDefinitionRhs, 259
pfDo, 260
pfDo?, 260
pfDoBody, 260
pfDocument, 244
pfEnSequence, 261
pfExit, 261
pfExit?, 261
pfExitCond, 261
pfExitExpr, 262
pfExport, 262
pfExpression, 262
pfFileName, 234
pfFirst, 262

pfFix, 263
pfFlattenApp, 239
pfForin, 264
pfForin?, 264
pfForinLhs, 265
pfForinWhole, 265
pfFree, 263
pfFree?, 263
pfFreeItems, 264
pfFromDom, 265
pfFromdom, 266
pfFromdom?, 266
pfFromdomDomain, 267
pfFromdomWhat, 266
pfGlobalLinePosn, 233
pfHide, 267
pfId, 244
pfId?, 244
pfIdPos, 244
pfIdSymbol, 245
pfIf, 267
pfIf?, 267
pfIfCond, 268
pfIfElse, 268
pfIfThen, 268
pfIfThenOnly, 268
pfImport, 269
pfInfApplication, 269
pfInline, 270
pfIterate, 269
pfIterate?, 269
pfLam, 270
pfLambda, 271
pfLambda2Sex, 315
pfLambda?, 271
pfLambdaArgs, 272
pfLambdaBody, 271
pfLambdaRets, 271
pfLambdaTran, 314
pfLeaf, 245
pfLeaf?, 245
pfLeafPosition, 246
pfLeafToken, 246
pfLhsRule2Sex, 316
pfLinePosn, 233
pfListOf, 243
pfLiteral2Sex, 302
pfLiteral?, 246
pfLiteralClass, 246
pfLiteralString, 247
pfLocal, 272
pfLocal?, 272
pfLocalItems, 273
pfLoop, 273
pfLoop1, 273
pfLoop?, 274
pfLoopIterators, 274
pfLp, 274
pfMacro, 275
pfMacro?, 275
pfMacroLhs, 275
pfMacroRhs, 275
pfMapParts, 234
pfMLambda, 276
pfMLambda?, 276
pfMLambdaArgs, 276
pfMLambdaBody, 277
pfname, 91
pfNoPosition, 412
pfNoPosition?, 410
pfNot?, 277
pfNotArg, 277
pfNothing, 243
pfNothing?, 243
pfNovalue, 277
pfNovalue?, 278
pfNovalueExpr, 278
pfOp2Sex, 306
pfOr, 278
pfOr?, 278
pfOrLeft, 279
pfOrRight, 279
pfParen, 279
pfParts, 247
pfPile, 247
pfPretend, 279
pfPretend?, 280
pfPretendExpr, 280
pfPretendType, 280
pfPushBody, 247
pfPushMacroBody, 241
pfQualType, 280

pfRestrict, 281
pfRestrict?, 281
pfRestrictExpr, 281
pfRestrictType, 281
pfRetractTo, 282
pfReturn, 282
pfReturn?, 282
pfReturnExpr, 282
pfReturnNoName, 283
pfReturnTyped, 283
pfRhsRule2Sex, 317
pfRule, 283
pfRule2Sex, 316
pfRule?, 284
pfRuleLhsItems, 284
pfRuleRhs, 284
pfSecond, 284
pfSequence, 285
pfSequence2Sex, 308
pfSequence2Sex0, 308
pfSequence?, 285
pfSequenceArgs, 285
pfSequenceToList, 236
pfSexpr, 248
pfSexpr,strip, 248
pfSourcePosition, 236
pfSourceStok, 241
pfSpread, 237
pfSuch, 242
pfSuchthat, 286
pfSuchThat2Sex, 305
pfSuchthat?, 286
pfSuchthatCond, 286
pfSymb, 249
pfSymbol, 249
pfSymbol?, 250
pfSymbolSymbol, 250
pfTagged, 286
pfTagged?, 287
pfTaggedExpr, 287
pfTaggedTag, 287
pfTaggedToTyped, 287
pfTaggedToTyped1, 242
pfTransformArg, 242
pfTree, 250
pfTuple, 290
pfTuple?, 290
pfTupleListOf, 290
pfTupleParts, 291
pfTweakIf, 288
pfTyped, 288
pfTyped?, 289
pfTypedId, 289
pfTypedType, 289
pfTyping, 289
pfUnSequence, 291
pfWDec, 291
pfWDeclare, 292
pfWhere, 292
pfWhere?, 292
pfWhereContext, 293
pfWhereExpr, 293
pfWhile, 293
pfWhile?, 294
pfWhileCond, 294
pfWith, 294
pfWrong, 294
pfWrong?, 295
phInterpret, 67
phIntReportMsgs, 66
phMacro, 219
phParse, 66
pileCforest, 338
pileColumn, 335
pileCtree, 338
pileForest, 336
pileForest1, 336
pileForests, 335
pilePlusComment, 334
pilePlusComments, 334
pileTree, 335
pmDontQuote?, 307
pname, 1039
poCharPosn, 375
poFileName, 358
poGetLineObject, 359
poGlobalLinePosn, 72
poLinePosn, 359
poNopos?, 358
poNoPosition, 412
poNoPosition?, 411
poPosImmediate?, 358

porigin, 88
posend, 126
posPointers, 376
ppos, 355
pquit, 630
pquitSpad2Cmd, 630
previousInterpreterFrame, 557
printAsTeX, 61
printLabelledList, 470
printStatisticsSummary, 57
printStorage, 58
printSynonyms, 470
printTypeAndTime, 58
printTypeAndTimeNormal, 59
printTypeAndTimeSaturn, 60
probeName, 974
processChPosesForOneLine, 374
processInteractive, 49
processInteractive1, 52
processKeyedError, 351
processMsgList, 367
processSynonymLine, 827
processSynonymLine,removeKeyFromLine, 826
processSynonyms, 33
protectedEVAL, 47
prTraceNames, 888
prTraceNames,fn, 888
pspacers, 849
ptimers, 849
punctuation?, 115
put, 1033
putDatabaseStuff, 363
putFTText, 377
putHist, 586
pvarPredTran, 320
qenum, 1040
qsquotient, 1052
qsremainder, 1052
queryClients, 506
queueUpErrors, 370
quit, 634
quitSpad2Cmd, 634
quotient2, 1073
random, 1073
rassocSub, 861
rdefinstream, 1059
rdefoutstream, 1060
rdigit?, 131
read, 638
readHiFi, 597
readSpad2Cmd, 638
readSpadProfileIfThere, 951
reclaim, 39
recordAndPrint, 56
recordFrame, 913
recordNewValue, 587
recordNewValue0, 587
recordOldValue, 588
recordOldValue0, 588
redundant, 372
remainder2, 1072
remFile, 355
remLine, 360
removeOption, 851
remover, 887
removeTracedMapSigs, 854
removeUndoLines, 923
rep, 369
replaceFile, 975
replaceSharps, 948
reportOperations, 807
reportOpsFromLisplib, 810
reportOpsFromLisplib0, 809
reportOpsFromLisplib1, 809
reportOpsFromUnitDirectly, 813
reportOpsFromUnitDirectly0, 813
reportOpsFromUnitDirectly1, 817
reportSpadTrace, 882
reportUndo, 917
reportWhatOptions, 931
reroot, 40
resetCounters, 848
resethashtables, 991
resetInCoreHist, 584
resetSpacers, 848
resetStackLimits, 21
resetTimers, 848
resetWorkspaceVariables, 646
restart0, 18
restoreHistory, 593
retract, 1049

rread, 601
ruleLhsTran, 320
rulePredicateTran, 317
runspad, 21
rwrite, 600
safeWritify, 602
sameMsg?, 374
sameUnionBranch, 61
satisfiesRegularExpressions, 933
satisfiesUserLevel, 447
saveDependentsHashTable, 1013
saveHistory, 591
saveMapSig, 843
savesystem, 642
saveUsersHashTable, 1014
sayBrightly1, 1043
sayExample, 535
sayKeyedMsg, 327
sayKeyedMsgLocal, 328
sayMSG, 329
sayMSG2File, 329
sayShowWarning, 818
scanCheckRadix, 131
scanCloser?, 123
scanComment, 113
scanDictCons, 134
scanError, 132
scanEsc, 120
scanEscape, 132
scanExponent, 124
scanIgnoreLine, 110
scanInsert, 135
scanKeyTableCons, 133
scanKeyTr, 117
scanNegComment, 114
scanNumber, 129
ScanOrPairVec, 612
ScanOrPairVec,ScanOrInner, 611
scanPossFloat, 118
scanPunCons, 137
scanPunct, 115
scanS, 128
scanSpace, 127
scanString, 127
scanToken, 111
scanTransform, 129
scanW, 125
scanWord, 123
search, 954
searchCurrentEnv, 954
searchTailEnv, 955
sec, 1089
sech, 1091
segmentKeyedMsg, 328
selectOption, 475
selectOptionLC, 475
separatePiles, 339
serverReadLine, 44
set, 799
set-restart-hook, 15
set1, 800
setCurrentLine, 41
setdatabase, 1000
setExpose, 688
setExposeAdd, 689
setExposeAddConstr, 692
setExposeAddGroup, 690
setExposeDrop, 693
setExposeDropConstr, 695
setExposeDropGroup, 694
setFortDir, 717
setFortPers, 748
setFortTmpDir, 715
setFunctionsCache, 699
setHistoryCore, 580
setInputLibrary, 659
setIOindex, 595
setLinkerArgs, 720
setMsgCatlessAttr, 363
setMsgForcedAttr, 362
setMsgForcedAttrList, 362
setMsgPrefix, 347
setMsgText, 347
setMsgUnforcedAttr, 365
setMsgUnforcedAttrList, 364
setNagHost, 746
setOutputAlgebra, 754
setOutputCharacters, 758
setOutputFormula, 781
setOutputFortran, 761
setOutputHtml, 772
setOutputLibrary, 656

setOutputMathml, 768
setOutputOpenMath, 777
setOutputTex, 788
setStreamsCalculate, 793
shortenForPrinting, 881
show, 806
showdatabase, 999
showInOut, 596
showInput, 595
showMsgPos?, 355
showSpad2Cmd, 806
shut, 972
size, 1039
spad, 20
spad-error-loc, 960
spad-long-error, 959
spad-save, 979
spad-short-error, 960
spad-syntax-error, 959
spad2BootCoerce, 1051
spadClosure?, 607
SpadInterpretStream, 27
spadReply, 885
spadReply,printName, 884
spadrread, 601
spadrwrite, 601
spadrwrite0, 600
spadStartUpMsgs, 19
spadTrace, 866
spadTrace,g, 865
spadTrace,isTraceable, 865
spadTraceAlias, 881
spadUntrace, 885
specialChar, 970
spleI, 119
spleI1, 120
splitIntoOptionBlocks, 443
squeeze, 1017
stackTraceOptionError, 851
startsComment?, 113
startsNegComment?, 114
streamChop, 72
StreamNull, 331
stringMatches?, 1075
StringToDir, 1076
stripLisp, 467
stripSpaces, 467
strpos, 1039
strposl, 1040
stupidIsSpadFunction, 896
subMatch, 116
substringMatch, 116
subTypes, 856
summary, 822
syGeneralErrorHere, 190
syIgnoredFromTo, 189
synonym, 824
synonymsForUserLevel, 825
synonymSpad2Cmd, 824
sySpecificErrorAtToken, 190
sySpecificErrorHere, 190
systemCommand, 444
tabbing, 360
terminateSystemCommand, 448
tersyscommand, 448
thefname, 91
theid, 90
theorigin, 88
thisPosIsEqual, 372
thisPosIsLess, 372
throwEvalTypeMsg, 909
To, 378
toFile?, 361
tokConstruct, 409
token-stack-show, 961
tokPart, 411
tokPosn, 411
tokTran, 463
tokType, 411
toScreen?, 349
trace, 837
trace1, 838
traceDomainConstructor, 870
traceDomainLocalOps, 870
tracelet, 894
traceOptionError, 847
traceReply, 889
traceSpad2Cmd, 837
trademark, 519
translateTrueFalse2YesNo, 651
translateYesNo2TrueFalse, 650
transOnlyOption, 850

transTraceItem, 853
unAbbreviateKeyword, 465
undo, 912
undoChanges, 589
undoCount, 919
undoFromFile, 590
undoInCore, 588
undoLocalModemapHack, 923
undoSingleStep, 921
undoSteps, 920
unescapeStringsInForm, 62
unifyStruct, 426
unifyStructVar, 427
unsqueeze, 1018
untrace, 852
untraceDomainConstructor, 873
untraceDomainConstructor,keepTraced?, 872
untraceDomainLocalOps, 870
untraceMapSubNames, 863
unwritable?, 602
updateCurrentInterpreterFrame, 555
updateDatabase, 1009
updateFromCurrentInterpreterFrame, 554
updateHist, 585
updateInCoreHist, 586
updateSourceFiles, 542
userLevelErrorMessage, 446
validateOutputDirectory, 716
vec2list, 1049
voidValue, 1047
what, 929
whatCommands, 931
whatConstructors, 935
whatSpad2Cmd, 930
whatSpad2Cmd,fixpat, 929
whichCat, 363
with, 937
workfiles, 939
workfilesSpad2Cmd, 939
wrap, 1037
write-browsedb, 1024
write-categorydb, 1025
write-compress, 1016
write-interpdb, 1022
write-operationdb, 1026
write-warmdata, 1027
writeHiFi, 598
writeHistModesAndValues, 599
writeInputLines, 583
writify, 606
writify,writifyInner, 603
writifyComplain, 602
xlCannotRead, 91
xlCmdBug, 98
xlConActive, 93
xlConsole, 94
xlConStill, 94
xlFileCycle, 92
xlIfBug, 97
xlIfSyntax, 96
xlMsg, 86
xlNoSuchFile, 90
xlOK, 86
xlOK1, 86
xlPrematureEOF, 86
xlPrematureFin, 95
xlSay, 89
xlSkip, 89
xlSkippingFin, 95
yesanswer, 533
zeroOneTran, 68
zsystemdevelopment, 943
zsystemdevelopment1, 944
zsystemDevelopmentSpad2Cmd, 943
defvar
allOperations, 991
allconstructors, 991
attributes, 1016
browse-stream, 990
browse-stream-stamp, 990
category-stream, 990
category-stream-stamp, 991
compress-stream, 989
compress-stream-stamp, 989
compressVectorLength, 988
compressvector, 988
defaultdomain-list, 987
eof, 24
hasCategory-hash, 987
interp-stream, 989
interp-stream-stamp, 989

miss, 988
monitor-domains, 1147
monitor-nrlibs, 1147
monitor-table, 1148
msghash, 325
operation-hash, 987
operation-stream, 989
operation-stream-stamp, 990
whitespace, 24
/editfile, 511
$nagMessages, 749
$reportBottomUpFlag, 738
$BreakMode, 653
$CommandSynonymAlist, 474
$EndServerSession, 43
$HTCompanionWindowID, 52
$HiFiAccess, 724
$InitialCommandSynonymAlist, 472
$InitialModemapFrame, 9
$InteractiveMode, 24
$NeedToSignalSessionManager, 44
$NonNullStream, 607
$NullStream, 607
$ProcessInteractiveValue, 52
$QuietCommand, 47
$RTspecialCharacters, 968
$SpadServerName, 12, 13
$SpadServer, 12
$UserLevel, 798
$abbreviateTypes, 752
$algebraFormat, 753
$algebraOutputFile, 753
$algebraOutputStream, 754
$attrCats, 362
$boot, 25
$cacheAlist, 699
$cacheMessages, 324
$clearExcept, 495
$clearOptions, 495
$compileDontDefineFunctions, 703
$compileRecurrence, 704
$constructors, 889
$current-directory, 7
$currentFrameNum, 43
$dalymode, 655
$defaultFortranType, 709
$defaultMsgDatabaseName, 8
$defaultSpecialCharacters, 965
$describeOptions, 524
$directory-list, 8
$displayDroppedMap, 730
$displayMsgNumber, 737
$displayOptions, 531
$displaySetValue, 739
$displayStartMsgs, 740
$domPvar, 49
$envHashTable, 953
$formulaFormat, 781
$formulaOutputFile, 781
$fortIndent, 706
$fortInts2Floats, 706
$fortLength, 707
$fortPersistence, 747
$fortranArrayStartingIndex, 713
$fortranDirectory, 717
$fortranFormat, 760
$fortranLibraries, 719
$fortranOptimizationLevel, 712
$fortranOutputFile, 760
$fortranPrecision, 709
$fortranSegment, 712
$fortranTmpDir, 714
$fractionDisplayType, 765
$frameAlist, 43
$frameMessages, 732
$frameNumber, 43
$frameRecord, 911
$fullScreenSysVars, 721
$functionTable, 498
$genValue, 53
$giveExposureWarning, 730
$globalExposureGroupAlist, 662
$highlightAllowed, 733
$historyDirectory, 575
$historyDisplayWidth, 722
$historyFileType, 575
$htmlFormat, 771
$htmlOutputFile, 771
$imPrGuys, 356
$imPrTagGuys, 365
$inputPromptType, 738
$intCoerceFailure, 32

$intRestart, 26
$intSpadReader, 32
$intTopLevel, 26
$interpOnly, 49
$lambdatype, 654
$library-directory-list, 9
$linearFormatScripts, 785
$linelength, 765
$localExposureDataDefault, 688
$localExposureData, 688
$margin, 765
$mathmlFormat, 767
$mathmlOutputFile, 767
$maximumFortranExpressionLength, 711
$minivectorNames, 49
$msgAlist, 324
$msgDatabaseName, 9, 10, 324
$msgdbNoBlanksAfterGroup, 326
$msgdbNoBlanksBeforeGroup, 326
$msgdbPrims, 325
$msgdbPunct, 325
$nagEnforceDouble, 750
$nagHost, 745
$nagMessages, 736
$ncMsgList, 27
$newcompErrorCount, 28
$noEvalTypeMsg, 909
$noParseCommands, 440
$noSubsumption, 952
$nopos, 28
$numericFailure, 1065
$oldBreakMode, 1065
$oldHistoryFileName, 574
$oldline, 445
$openMathFormat, 776
$openMathOutputFile, 776
$openServerIfTrue, 10
$optionAlist, 836
$pfMacros, 98
$plainRTspecialCharacters, 968
$plainSpecialCharacters0, 966
$plainSpecialCharacters1, 966
$plainSpecialCharacters2, 967
$plainSpecialCharacters3, 967
$preLength, 353
$prettyprint, 797
$previousBindings, 911
$printAnyIfTrue, 726
$printFortranDecs, 708
$printLoadMsgs, 727
$printMsgsToFile, 731
$printStatisticsSummaryIfTrue, 741
$printTimeIfTrue, 742
$printTypeIfTrue, 743
$printVoidIfTrue, 744
$promptMsg, 28
$quitCommandType, 791
$quitTag, 20
$relative-directory-list, 10, 11
$relative-library-directory-list, 11
$repGuys, 373
$reportBottomUpFlag, 728
$reportCoerceIfTrue, 729
$reportCompilation, 795
$reportInstantiations, 734
$reportInterpOnly, 735
$reportOptimization, 796
$reportSpadTrace, 836
$reportUndo, 912
$setOptionNames, 799
$sockBufferLength, 44
$spad-errors, 957
$spadroot, 11, 12
$specialCharacterAlist, 969
$specialCharacters, 969
$streamCount, 792
$streamsShowAll, 794
$syscommands, 440
$systemCommands, 439
$testingErrorPrefix, 325
$testingSystem, 742
$texFormatting, 325
$texFormat, 787
$texOutputFile, 787
$toWhereGuys, 361
$tokenCommands, 471
$traceNoisely, 836
$traceOptionList, 836
$tracedMapSignatures, 836
$underbar, 582
$undoFlag, 911
$useEditorForShowOutput, 786

$useFullScreenHelp, 723
$useInternalHistoryTable, 575
$useIntrinsicFunctions, 710
$whatOptions, 929
boot-line-stack, 951
credits, 1
curinstream, 23
curoutstream, 23
dotdot, 321
echo-meta, 952
Else?, 80
Elseif?, 79
ElseifKeepPart, 78
ElseifSkipPart, 78
ElseifSkipToEnd, 78
ElseKeepPart, 79
ElseSkipToEnd, 78
errorinstream, 23
erroroutstream, 24
file-closed, 952
If?, 79
IfKeepPart, 77
IfSkipPart, 78
IfSkipToEnd, 77
in-stream, 951
incCommands, 98
infgeneric, 107
input-libraries, 660
KeepPart?, 80
line-handler, 957
npPParg, 202, 207
out-stream, 952
output-library, 657
scanCloser, 122
scanDict, 134
scanKeyTable, 133
scanKeyWords, 105
scanPun, 136
SkipEnd?, 80
SkipPart?, 81
Skipping?, 81
StreamNil, 104
Top, 77
Top?, 79
xtokenreader, 958
delasc
calledby spadUntrace, 886
Delay, 103
calledby incAppend, 87
calledby incIgen, 75
calledby incLude, 76
calledby incRgen, 103
calledby incZip, 74
calledby next, 38
defun, 103
deldatabase, 1001
calledby abbreviationsSpad2Cmd, 479
defun, 1001
delete
calledby breaklet, 895
calledby setExposeAddConstr, 692
calledby setExposeDropConstr, 695
calledby setExposeDropGroup, 694
calledby trace1, 838
calledby tracelet, 894
calledby untraceDomainConstructor, 873
calledby workfilesSpad2Cmd, 939
deleteAssoc
calledby clearCmdParts, 501
deleteFile, 1036
calledby clearCmdAll, 499
calls erase, 1036
calls pathname, 1037
uses $erase, 1037
defun, 1036
delt, 1067
defmacro, 1067
describe, 524
calls describespad2cmd, 524
defun, 524
describe help page, 523
manpage, 523
describeFortPersistence, 748
calledby setFortPers, 748
calls sayBrightly, 748
uses $fortPersistence, 748
defun, 748
describeInputLibraryArgs, 660
calledby setInputLibrary, 659
calls sayBrightly, 660
defun, 660
describeOutputLibraryArgs, 657

calledby setOutputLibrary, 656
calls sayBrightly, 657
defun, 657
describeSetFortDir, 718
calledby setFortDir, 717
calls sayBrightly, 718
uses $fortranDirectory, 718
defun, 718
describeSetFortTmpDir, 716
calledby setFortTmpDir, 715
calls sayBrightly, 716
uses $fortranTmpDir, 716
defun, 716
describeSetLinkerArgs, 720
calledby setLinkerArgs, 720
calls sayBrightly, 720
uses $fortranLibraries, 720
defun, 720
describeSetNagHost, 746
calledby setNagHost, 746
calls sayBrightly, 746
uses $nagHost, 746
defun, 746
describeSetOutputAlgebra, 756
calledby setOutputAlgebra, 754
calls sayBrightly, 756
calls setOutputAlgebra, 756
defun, 756
describeSetOutputFormula, 784
calledby setOutputFormula, 782
calls sayBrightly, 784
calls setOutputFormula, 784
defun, 784
describeSetOutputFortran, 763
calledby setOutputFortran, 761
calls sayBrightly, 763
calls setOutputFortran, 763
defun, 763
describeSetOutputHtml, 774
calledby setOutputHtml, 772
calls sayBrightly, 774
calls setOutputHtml, 774
defun, 774
describeSetOutputMathml, 770
calledby setOutputMathml, 768
calls sayBrightly, 770
calls setOutputMathml, 770
defun, 770
describeSetOutputOpenMath, 779
calledby setOutputOpenMath, 777
calls sayBrightly, 779
calls setOutputOpenMath, 779
defun, 779
describeSetOutputTex, 790
calledby setOutputTex, 788
calls sayBrightly, 790
calls setOutputTex, 790
defun, 790
describeSetStreamsCalculate, 793
calledby setStreamsCalculate, 793
calls sayKeyedMsg, 793
uses $streamCount, 794
defun, 793
describeSpad2Cmd, 524
calls cleanline, 525
calls flatten, 525
calls getdatabase, 525
calls sayMessage, 525
calls selectOptionLC, 525
uses $EmptyEnvironment, 525
uses $describeOptions, 525
uses $e, 525
defun, 524
describespad2cmd
calledby describe, 524
desiredMsg, 350
calledby ncHardError, 350
calledby ncSoftError, 349
defun, 350
devaluate
calledby /tracereply, 884
calledby ?t, 893
calledby addTraceItem, 892
calledby mkEvalable, 903
calledby prTraceNames,fn, 888
calledby printTypeAndTimeSaturn, 60
calledby shortenForPrinting, 881
calledby spadReply,printName, 884
calledby spadUntrace, 885
calledby trace1, 838
calledby transTraceItem, 853

calledby untraceDomainConstructor,keepTrace?, 872
calledby writify,writifyInner, 603
dewritify, 611
calledby SPADRREAD, 601
calls ScanOrPairVec, 611
calls dewritify,dewritifyInner, 611
calls function, 611
uses $seen, 611
defun, 611
dewritify,dewritifyInner, 608
calledby dewritify,dewritifyInner, 608
calledby dewritify, 611
calls concat, 608
calls dewritify,dewritifyInner, 608
calls error, 608
calls exit, 608
calls gensymmer, 608
calls hget, 608
calls hput, 608
calls intp, 608
calls make-instream, 608
calls poundsign, 608
calls qcar, 608
calls qcdr, 608
calls qrplaca, 608
calls qrplacd, 608
calls qsetvelt, 608
calls qvelt, 608
calls qvmaxindex, 608
calls seq, 608
calls spaddifference, 608
calls vecp, 608
calls vmread, 608
uses $NonNullStream, 608
uses $NullStream, 608
uses $seen, 608
defun, 608
DFAcos, 1084
defmacro, 1084
DFAcosh, 1087
defmacro, 1087
DFAdd, 1080
defmacro, 1080
DFAsin, 1084
defmacro, 1084
DFAsinh, 1086
defmacro, 1086
DFAtan, 1085
defmacro, 1085
DFAtan2, 1085
defmacro, 1085
DFAtanh, 1087
defmacro, 1087
DFCos, 1084
defmacro, 1084
DFCosh, 1086
defmacro, 1086
DFDivide, 1081
defmacro, 1081
DFEql, 1081
defmacro, 1081
DFExp, 1083
defmacro, 1083
DFExpt, 1083
defmacro, 1083
DFIntegerDivide, 1082
defmacro, 1082
DFIntegerExpt, 1083
defmacro, 1083
DFIntegerMultiply, 1080
defmacro, 1080
DFLessThan, 1079
defmacro, 1079
DFLog, 1082
defmacro, 1082
DFLogE, 1082
defmacro, 1082
DFMax, 1081
defmacro, 1081
DFMin, 1081
defmacro, 1081
DFMinusp, 1079
defmacro, 1079
DFMultiply, 1080
defmacro, 1080
DFSin, 1083
defmacro, 1083
DFSinh, 1085
defmacro, 1085
DFSqrt, 1082
defmacro, 1082

DFSubtract, 1080
defmacro, 1080
DFTan, 1084
defmacro, 1084
DFTanh, 1086
defmacro, 1086
DFUnaryMinus, 1079
defmacro, 1079
DFZerop, 1079
defmacro, 1079
diffAlist, 914
calledby recordFrame, 913
calls assoc, 915
calls assq, 915
calls exit, 915
calls lassq, 915
calls reportUndo, 915
calls seq, 915
calls tmp1, 915
defun, 914
dig2fix
calledby isSharpVarWithNum, 876
digit?, 119
calledby scanExponent, 124
calledby scanPossFloat, 118
calledby scanToken, 111
calls digitp, 119
defun, 119
digitp, 1039
calledby digit?, 119
calledby digitp, 1039
calledby isSharpVarWithNum, 876
calledby isgenvar, 877
calls digitp, 1039
defun, 1039
DirToString, 1076
calledby fnameDirectory, 1076
defun, 1076
disableHist, 599
calledby fetchOutput, 596
calledby historySpad2Cmd, 578
calledby restoreHistory, 593
calledby setHistoryCore, 581
calledby showInOut, 596
calledby showInput, 595
calledby undoFromFile, 590
calledby undoInCore, 589
calledby updateHist, 585
calls histFileErase, 599
calls histFileName, 599
uses $HiFiAccess, 599
defun, 599
display, 531
calls displayspad2cmd, 531
defun, 531
display help page, 529
manpage, 529
displayCondition, 461
calledby displayProperties, 457
calls bright, 461
calls concat, 461
calls pred2English, 461
calls sayBrightly, 461
defun, 461
displayExposedConstructors, 696
calledby setExposeAddConstr, 692
calledby setExposeAddGroup, 690
calledby setExposeAdd, 689
calledby setExposeDropConstr, 695
calledby setExposeDropGroup, 694
calledby setExpose, 689
calls centerAndHighlight, 696
calls sayKeyedMsg, 696
uses $localExposureData, 696
defun, 696
displayExposedGroups, 696
calledby setExposeAddGroup, 690
calledby setExposeAdd, 689
calledby setExposeDropGroup, 694
calledby setExpose, 688
calls centerAndHighlight, 696
calls sayKeyedMsg, 696
uses $interpreterFrameName, 696
uses $localExposureData, 696
defun, 696
displayFrameNames, 559
calledby frameSpad2Cmd, 562
calls bright, 559
calls framename, 559
calls sayKeyedMsg, 559
uses $interpreterFrameRing, 559
defun, 559

displayHiddenConstructors, 697
calledby setExposeAddGroup, 690
calledby setExposeDropConstr, 695
calledby setExposeDropGroup, 694
calledby setExposeDrop, 693
calledby setExpose, 689
calls centerAndHighlight, 697
calls sayKeyedMsg, 697
uses $localExposureData, 697
defun, 697
displayMacro, 449
calledby displayMacros, 534
calledby displayProperties, 458
calls bright, 450
calls isInterpMacro, 449
calls mathprint, 450
calls object2String, 450
calls sayBrightly, 450
calls strconc, 450
uses $op, 450
defun, 449
displayMacros, 534
calledby displaySpad2Cmd, 531
calls displayMacro, 534
calls displayParserMacro, 534
calls exit, 534
calls getInterpMacroNames, 534
calls getParserMacroNames, 534
calls member, 534
calls remdup, 534
calls sayBrightly, 534
calls seq, 534
defun, 534
displayMode, 462
calledby displayProperties, 457
calls bright, 462
calls concat, 462
calls fixObjectForPrinting, 462
calls prefix2String, 463
calls sayBrightly, 462
defun, 462
displayModemap, 462
calledby displayProperties, 458
calls bright, 462
calls concat, 462
calls formatSignature, 462
calls sayBrightly, 462
defun, 462
displayOperations, 533
calledby displaySpad2Cmd, 531
calls reportOpSymbol, 533
calls sayKeyedMsg, 533
calls yesanswer, 533
defun, 533
displayOperationsFromLisplib, 812
calledby reportOpsFromLisplib, 810
calls centerAndHighlight, 812
calls eqsubstlist, 812
calls formatOperationAlistEntry, 813
calls getdatabase, 812
calls msort, 812
calls remdup, 812
calls reportOpsFromUnitDirectly, 812
calls say2PerLine, 813
calls specialChar, 812
uses $FormalMapVariableList, 813
uses $linelength, 813
defun, 812
displayParserMacro, 460
calledby displayMacros, 534
calledby displayProperties, 458
calls pfPrintSrcLines, 460
uses $pfMacros, 460
defun, 460
displayProperties, 457
calledby displaySpad2Cmd, 531
calls bright, 458
calls displayCondition, 457
calls displayMacro, 458
calls displayModemap, 458
calls displayMode, 457
calls displayParserMacro, 458
calls displayProperties,sayFunctionDeps, 458
calls displayType, 457
calls displayValue, 457
calls exit, 458
calls fixObjectForPrinting, 458
calls getAndSay, 457
calls getIProplist, 457
calls getInterpMacroNames, 457
calls getI, 457

calls getParserMacroNames, 457
calls getWorkspaceNames, 457
calls interpFunctionDepAlists, 457
calls isInternalMapName, 457
calls isInterpMacro, 458
calls member, 458
calls msort, 457
calls prefix2String, 458
calls qcar, 457
calls qcdr, 457
calls remdup, 457
calls sayKeyedMsg, 457
calls sayMSG, 458
calls seq, 458
calls terminateSystemCommand, 458
uses $dependeeAlist, 458
uses $dependentAlist, 458
uses $frameMessages, 458
uses $interpreterFrameName, 458
defun, 457
displayProperties,sayFunctionDeps, 452
calledby displayProperties, 458
calls bright, 454
calls exit, 454
calls getalist, 454
calls sayMSG, 454
calls seq, 454
uses $dependeeAlist, 454
uses $dependentAlist, 454
defun, 452
displayRule
calledby displayValue, 455
displaySetOptionInformation, 647
calledby set1, 800
calls boot-equal, 647
calls bright, 647
calls centerAndHighlight, 647
calls concat, 647
calls displaySetVariableSettings, 647
calls eval, 647
calls literals, 647
calls object2String, 647
calls sayBrightly, 647
calls sayMSG, 647
calls sayMessage, 647
calls specialChar, 647
calls translateTrueFalse2YesNo, 647
uses $linelength, 647
defun, 647
displaySetVariableSettings, 649
calledby displaySetOptionInformation, 647
calledby set1, 800
calls bright, 649
calls centerAndHighlight, 649
calls concat, 649
calls eval, 649
calls fillerSpaces, 649
calls literals, 649
calls object2String, 649
calls poundsign, 649
calls satisfiesUserLevel, 649
calls sayBrightly, 649
calls say, 649
calls spaddifference, 649
calls specialChar, 649
calls translateTrueFalse2YesNo, 649
calls tree, 649
uses $linelength, 649
defun, 649
displaySpad2Cmd
calls abbQuery, 531
calls displayMacros, 531
calls displayOperations, 531
calls displayProperties, 531
calls displayWorkspaceNames, 531
calls listConstructorAbbreviations, 531
calls opOf, 531
calls sayMessage, 532
calls selectOptionLC, 532
uses $EmptyEnvironment, 532
uses $displayOptions, 532
uses $e, 532
displayspad2cmd
calledby display, 531
displayType, 456
calledby displayProperties, 457
calls concat, 456
calls fixObjectForPrinting, 456
calls objMode, 456
calls pname, 456
calls prefix2String, 456
calls sayMSG, 456

uses $op, 456
defun, 456
displayValue, 455
calledby displayProperties, 457
calls concat, 455
calls displayRule, 455
calls fixObjectForPrinting, 455
calls form2String, 455
calls getdatabase, 455
calls mathprint, 455
calls objMode, 455
calls objValUnwrap, 455
calls outputFormat, 455
calls pname, 455
calls prefix2String, 455
calls sayMSG, 455
calls strconc, 455
uses $EmptyMode, 455
uses $op, 455
defun, 455
displayWorkspaceNames, 450
calledby displaySpad2Cmd, 531
calls getInterpMacroNames, 450
calls getParserMacroNames, 450
calls getWorkspaceNames, 450
calls msort, 450
calls sayAsManyPerLineAsPossible, 450
calls sayBrightly, 450
calls sayMessage, 450
calls setdifference, 450
defun, 450
divide2, 1072
defun, 1072
dlen, 1066
defmacro, 1066
domainsOf
calledby make-databases, 1010
domainToGenvar, 851
calledby getTraceOption,hn, 844
calledby transTraceItem, 853
calls evalDomain, 851
calls genDomainTraceName, 851
calls getdatabase, 851
calls opOf, 851
calls unabbrevAndLoad, 851
uses $doNotAddEmptyModeIfTrue, 851
defun, 851
domArg, 420
calledby hasCaty, 418
calledby mkDomPvar, 430
local ref $FormalMapVariableList, 420
defun, 420
domArg2, 420
calledby hasCaty, 418
calls isSharpVar, 420
calls subCopy, 420
local ref $domPvar, 420
defun, 420
done
calledby insertPos, 377
doSystemCommand, 442
calledby ExecuteInterpSystemCommand, 33
calls concat, 442
calls expand-tabs, 442
calls getFirstWord, 442
calls handleNoParseCommands, 442
calls handleParsedSystemCommands, 442
calls handleTokensizeSystemCommands, 442
calls member, 442
calls processSynonyms, 442
calls splitIntoOptionBlocks, 442
calls substring, 442
calls unAbbreviateKeyword, 442
uses $noParseCommands, 442
uses $tokenCommands, 442
uses line, 442
defun, 442
dotdot, 321
defvar, 321
downcase
calledby apropos, 935
calledby selectOptionLC, 475
calledby set1, 800
calledby setOutputCharacters, 758
calledby whatSpad2Cmd,fixpat, 929
dqAppend, 342
calledby dqConcat, 341
calledby pileCtree, 338
defun, 342
dqConcat, 341

calledby dqConcat, 341
calledby enPile, 338
calledby separatePiles, 339
calls dqAppend, 341
calls dqConcat, 341
defun, 341
dqToList, 342
calledby intloopEchoParse, 69
calledby ncloopParse, 38
defun, 342
dqUnit, 341
calledby enPile, 338
calledby lineoftoks, 109
calledby scanToken, 112
calledby separatePiles, 339
defun, 341
drop[9]
called by frameSpad2Cmd, 562
dropInputLibrary, 661
calledby addInputLibrary, 660
calledby openOutputLibrary, 658
calledby setInputLibrary, 659
uses input-libraries, 661
defun, 661
dropLeadingBlanks
calledby processSynonymLine,removeKeyFromLine, 826
dsetaref2, 1069
defmacro, 1069
dsetelt, 1067
defmacro, 1067
dumbTokenize, 463
calledby handleParsedSystemCommands, 464
calledby handleTokensizeSystemCommands, 443
calledby parseSystemCmd, 465
calls stripSpaces, 463
defun, 463

echo-meta, 952
defvar, 952
edit, 540
calls editSpad2Cmd, 540
defun, 540
edit help page, 539
manpage, 539
editFile, 541
calledby editSpad2Cmd, 540
calledby reportOpsFromLisplib1, 809
calledby reportOpsFromUnitDirectly1, 817
calls namestring, 541
calls obey, 541
calls pathname, 541
calls strconc, 541
defun, 541
editSpad2Cmd, 540
calledby edit, 540
calls $FINDFILE, 540
calls editFile, 540
calls pathnameDirectory, 540
calls pathnameName, 540
calls pathnameType, 540
calls pathname, 540
calls updateSourceFiles, 540
uses /editfile, 541
defun, 540
Else?, 80
calledby xlIfSyntax, 96
calls QUOTIENT, 80
defvar, 80
ElseIf?, 79
calledby incLude1, 82
calls QUOTIENT, 79
defvar, 79
ElseifKeepPart, 78
defvar, 78
ElseifSkipPart, 78
defvar, 78
ElseifSkipToEnd, 78
defvar, 78
ElseKeepPart, 79
defvar, 79
ElseSkipToEnd, 78
defvar, 78
elt32, 1048
defmacro, 1048
embed
calledby traceDomainConstructor, 871
emptyInterpreterFrame, 552
calledby addNewInterpreterFrame, 557

calledby initializeInterpreterFrameRing, 551
uses $HiFiAccess, 552
uses $HistListAct, 552
uses $HistListLen, 552
uses $HistList, 552
uses $HistRecord, 552
uses $localExposureDataDefault, 552
defun, 552
enable-backtrace
calledby break, 896
calledby ncBug, 366
enPile, 338
calledby pileCforest, 338
calls dqConcat, 338
calls dqUnit, 338
calls firstTokPosn, 338
calls lastTokPosn, 338
calls tokConstruct, 338
defun, 338
entryWidth
calledby printLabelledList, 470
eof
usedby fin, 544
eofp, 973
defun, 973
eqcar
calledby StreamNull, 331
calledby hasCateSpecialNew, 433
calledby hasCateSpecial, 432
calledby pfAbSynOp?, 410
calledby poNoPosition?, 411
eqpileTree, 337
calledby pileForest1, 336
calls npNull, 337
calls pileColumn, 337
calls pileForests, 337
defun, 337
eqsubstlist
calledby displayOperationsFromLisplib, 812
calledby reportOpsFromLisplib, 810
erase
calledby deleteFile, 1036
calledby reportOpsFromLisplib1, 809
calledby reportOpsFromUnitDirectly1, 817
calledby saveDependentsHashTable, 1013
calledby saveUsersHashTable, 1014
erMsgCompare, 368
calls compareposns, 368
calls getMsgPos, 368
defun, 368
erMsgSep, 368
calledby erMsgSort, 367
calls getMsgPos, 368
calls poNopos?, 368
defun, 368
erMsgSort, 367
calledby processMsgList, 367
calls erMsgSep, 367
calls listSort, 367
defun, 367
error
calledby basicLookupCheckDefaults, 1063
calledby basicLookup, 1061
calledby charDigitVal, 613
calledby dewritify,dewritifyInner, 608
calledby gensymInt, 612
calledby myWritable?, 1078
errorinstream, 23
defvar, 23
erroroutstream, 24
defvar, 24
eval
calledby NRTevalDomain, 1064
calledby displaySetOptionInformation, 647
calledby displaySetVariableSettings, 649
calledby evalDomain, 903
evalCategory, 949
calledby evaluateType1, 908
calls isPartialMode, 949
calls ofCategory, 949
defun, 949
evalDomain, 903
calledby NRTevalDomain, 1064
calledby domainToGenvar, 851
calledby reportOpsFromUnitDirectly, 814
calls concat, 903
calls eval, 903
calls mkEvalable, 903
calls prefix2String, 903
calls sayMSG, 903

calls startTimingProcess, 903
calls stopTimingProcess, 903
local ref $evalDomain, 903
defun, 903
evaluateSignature, 910
calledby evaluateType, 906
calls evaluateType, 910
defun, 910
evaluateType, 906
calledby evaluateSignature, 910
calledby evaluateType1, 908
calledby evaluateType, 906
calledby reportOperations, 808
calls bottomUp, 906
calls constructor?, 906
calls evaluateSignature, 906
calls evaluateType, 906
calls getValue, 906
calls isDomainValuedVariable, 906
calls member, 906
calls mkAtree, 906
calls objVal, 906
calls qcar, 906
calls qcdr, 906
calls throwEvalTypeMsg, 906
local def $expandSegments, 906
local ref $EmptyMode, 906
defun, 906
evaluateType1, 907
calls bottumUp, 908
calls categoryForm?, 907
calls coerceOrRetract, 908
calls constructor?, 907
calls evalCategory, 908
calls evaluateType, 908
calls getAndEvalConstructorArgument, 908
calls getConstructorSignature, 907
calls getdatabase, 908
calls makeOrdinal, 908
calls mkAtree, 908
calls objValUnwrap, 908
calls putTarget, 908
calls qcar, 908
calls qcdr, 908
calls replaceSharps, 907
calls throwEvalTypeMsg, 907
calls throwKeyedMsgCannotCoerceWithValue, 908
local ref $EmptyMode, 908
local ref $quadSymbol, 908
defun, 907
ExecuteInterpSystemCommand, 33
calledby InterpExecuteSpadSystemCommand, 32
calls doSystemCommand, 33
calls intProcessSynonyms, 33
calls substring, 33
uses $currentLine, 33
defun, 33
executeQuietCommand, 47
calledby serverReadLine, 44
calls make-string, 47
calls parseAndInterpret, 47
calls sockGetString, 47
uses $MenuServer, 47
uses $QuietCommand, 47
catches, 47
defun, 47
exit
calledby /tracereply, 884
calledby abbreviationsSpad2Cmd, 479
calledby apropos, 935
calledby clearCmdParts, 501
calledby coerceSpadArgs2E, 855
calledby dewritify,dewritifyInner, 608
calledby diffAlist, 915
calledby displayMacros, 534
calledby displayProperties,sayFunctionDeps, 454
calledby displayProperties, 458
calledby flattenOperationAlist, 873
calledby funfind,LAM, 864
calledby getAliasIfTracedMapParameter, 879
calledby getBpiNameIfTracedMap, 880
calledby getPreviousMapSubNames, 860
calledby getTraceOption,hn, 843
calledby getTraceOptions, 842
calledby getTraceOption, 844
calledby getWorkspaceNames, 451
calledby historySpad2Cmd, 579
calledby importFromFrame, 559

calledby isListOfIdentifiersOrStrings, 859
calledby isListOfIdentifiers, 858
calledby orderBySlotNumber, 883
calledby prTraceNames,fn, 888
calledby prTraceNames, 888
calledby recordFrame, 913
calledby removeUndoLines, 923
calledby reportUndo, 918
calledby runspad, 21
calledby set1, 800
calledby spadReply,printName, 884
calledby spadReply, 885
calledby spadTrace,isTraceable, 866
calledby spadTrace, 866
calledby spadUntrace, 886
calledby subTypes, 856
calledby trace1, 838
calledby traceDomainConstructor, 871
calledby traceReply, 889
calledby undoFromFile, 590
calledby undoLocalModemapHack, 923
calledby undoSingleStep, 921
calledby untraceDomainConstructor,keepTraced, 872
calledby untraceDomainConstructor, 873
calledby whatConstructors, 935
calledby whatSpad2Cmd, 930
calledby writify,writifyInner, 603

expand-tabs
calledby doSystemCommand, 442
calledby incLude1, 82

extendLocalLibdb
calledby library, 1005

fbpip
calledby mkEvalable, 903

fetchKeyedMsg, 326
calledby getKeyedMsg, 327
calls cacheKeyedMsg, 326
calls object2Identifier, 326
uses *msghash*, 326
uses $defaultMsgDatabaseName, 326
defun, 326

fetchOutput, 596
calls assq, 596
calls boot-equal, 596
calls disableHist, 596
calls getI, 596
calls readHiFi, 596
calls spaddifference, 596
calls throwKeyedMsg, 596
defun, 596

file-closed, 952
usedby init-boot/spad-reader, 958
defvar, 952

filep
calledby setOutputLibrary, 656

fillerSpaces, 20
calledby displaySetVariableSettings, 649
calledby justifyMyType, 62
calledby printLabelledList, 470
calledby spadStartUpMsgs, 19
calls ifcar, 20
defun, 20

filterAndFormatConstructors, 934
calledby whatSpad2Cmd, 930
calls blankList, 934
calls centerAndHighlight, 934
calls filterListOfStringsWithFn, 934
calls function, 934
calls pp2Cols, 934
calls sayMessage, 934
calls specialChar, 934
calls whatConstructors, 934
uses $linelength, 934
defun, 934

filterListOfStrings, 932
calledby apropos, 935
calledby whatCommands, 932
calls satisfiesRegularExpressions, 932
defun, 932

filterListOfStringsWithFn, 933
calledby filterAndFormatConstructors, 934
calledby printSynonyms, 470
calls satisfiesRegularExpressions, 933
defun, 933

fin, 544
uses eof, 544
defun, 544
throws, 544

fin help page, 543
manpage, 543

findfile
calledby readSpad2Cmd, 638
findFrameInRing, 555
calledby changeToNamedInterpreterFrame, 556
calls boot-equal, 555
calls frameName, 555
uses $interpreterFrameRing, 555
defun, 555
firstTokPosn, 339
calledby enPile, 338
calls tokPosn, 339
defun, 339
fixObjectForPrinting, 452
calledby clearCmdParts, 501
calledby displayMode, 462
calledby displayProperties, 458
calledby displayType, 456
calledby displayValue, 455
calls member, 452
calls object2Identifier, 452
calls pname, 452
calls strconc, 452
uses $msgdbPrims, 452
defun, 452
flatten, 527
calledby describeSpad2Cmd, 525
defun, 527
flattenOperationAlist, 873
calledby spadTrace, 866
calls exit, 873
calls seq, 873
defun, 873
float
calledby ptimers, 849
float2Sex, 303
calledby pfLiteral2Sex, 302
uses $useBFasDefault, 303
defun, 303
flowSegmentedMsg
calledby msgOutputter, 351
calledby msgText, 62
calledby sayKeyedMsgLocal, 328
calledby traceReply, 889
flung
usedby intloopSpadProcess, 65
fnameDirectory, 1076
calledby myWritable?, 1078
calls DirToString, 1076
defun, 1076
fnameExists?, 1077
calledby myWritable?, 1078
defun, 1077
fnameMake, 1075
calledby fnameNew, 1078
calls StringToDir, 1075
defun, 1075
fnameName, 1077
defun, 1077
fnameNew, 1078
calls fnameMake, 1078
defun, 1078
fnameReadable?, 1077
defun, 1077
fnameType, 1077
defun, 1077
fnameWritable?, 1078
calls myWriteable?, 1078
defun, 1078
for, 389
syntax, 389
form2String
calledby displayValue, 455
calledby reportOpsFromLisplib, 810
form2StringAsTeX
calledby printTypeAndTimeSaturn, 60
form2StringWithWhere
calledby reportOpsFromLisplib, 810
formatAttribute
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
formatOperation
calledby reportOpsFromUnitDirectly, 814
formatOperationAlistEntry
calledby displayOperationsFromLisplib, 813
formatOpType
calledby reportOpsFromUnitDirectly, 814
formatSignature
calledby compiledLookupCheck, 498
calledby displayModemap, 462
frame, 561

calls frameSpad2Cmd, 561
defun, 561
frame help page, 545
manpage, 545
frameEnvironment, 552
calledby importFromFrame, 559
calls frameInteractive, 552
defun, 552
frameInteractive
calledby frameEnvironment, 552
frameName, 548
calledby findFrameInRing, 555
defun, 548
framename
calledby addNewInterpreterFrame, 557
calledby closeInterpreterFrame, 558
calledby displayFrameNames, 559
calledby importFromFrame, 559
frameNames, 552
calledby importFromFrame, 559
uses $interpreterFrameRing, 552
defun, 552
frameSpad2Cmd, 562
calledby frame, 561
calls addNewInterpreterFrame, 562
calls closeInterpreterFrame, 562
calls displayFrameNames, 562
calls drop[9], 562
calls helpSpad2Cmd, 562
calls importFromFrame, 562
calls import, 562
calls last, 562
calls names, 562
calls new, 562
calls nextInterpreterFrame, 562
calls next, 562
calls object2Identifier, 562
calls previousInterpreterFrame, 562
calls qcar, 562
calls qcdr, 562
calls selectOptionLC, 562
calls throwKeyedMsg, 562
uses $options, 562
defun, 562
From, 378
calledby syIgnoredFromTo, 189
defun, 378
FromTo, 378
calledby syIgnoredFromTo, 189
defun, 378
function
calledby dewritify, 611
calledby filterAndFormatConstructors, 934
calledby isSystemDirectory, 1030
calledby writify, 606
functionp, 1041
calls identp, 1041
defun, 1041
funfind, 864
defmacro, 864
funfind,LAM, 864
calls SEQ, 864
calls exit, 864
calls isFunctor, 864
calls qcar, 864
defun, 864

gbc-time
calledby statisticsInitialization, 1029
genCategoryTable
calledby write-categorydb, 1025
genDomainTraceName, 852
calledby domainToGenvar, 851
calls genvar, 852
calls lassoc, 852
uses $domainTraceNameAssoc, 852
defun, 852
gensymInt, 612
calls charDigitVal, 612
calls error, 612
calls gensymp, 612
calls pname, 612
defun, 612
gensymmer
calledby dewritify,dewritifyInner, 608
gensymp
calledby breaklet, 895
calledby gensymInt, 612
calledby isTraceGensym, 865
calledby letPrint2, 877
calledby letPrint3, 878
calledby letPrint, 875

calledby spadTrace,isTraceable, 866
calledby tracelet, 894
genvar
calledby genDomainTraceName, 852
get
calledby ?t, 892
calledby augmentTraceNames, 862
calledby clearCmdParts, 501
calledby getAliasIfTracedMapParameter, 879
calledby getBpiNameIfTracedMap, 880
calledby getMapSig, 843
calledby getMapSubNames, 859
calledby getPreviousMapSubNames, 860
calledby importFromFrame, 559
calledby isDomainValuedVariable, 949
calledby isInterpOnlyMap, 862
calledby isUncompiledMap, 861
calledby putHist, 586
calledby restoreHistory, 593
calledby transTraceItem, 853
calledby writeHistModesAndValues, 599
get-current-directory, 36
calledby restart, 17
defun, 36
getAliasIfTracedMapParameter, 879
calledby mapLetPrint, 874
calls exit, 879
calls get, 879
calls isSharpVarWithNum, 879
calls pname, 880
calls seq, 880
calls spaddifference, 879
calls string2pint-n, 879
calls substring, 879
uses $InteractiveFrame, 880
defun, 879
getalist
calledby displayProperties,sayFunctionDeps, 454
calledby importFromFrame, 559
calledby interpFunctionDepAlists, 461
calledby isExposedConstructor, 812
calledby setExposeAddGroup, 691
calledby setExposeDropGroup, 694
getAndEvalConstructorArgument, 948
calledby evaluateType1, 908
calls compFailure, 948
calls getValue, 948
calls isLocalVar, 948
calls isWrapped, 948
calls objMode, 948
calls objNewWrap, 948
calls objVal, 948
calls timedEVALFUN, 948
defun, 948
getAndSay, 457
calledby displayProperties, 457
calls getI, 457
calls sayMSG, 457
defun, 457
getArgValue
calledby interpret1, 54
getBpiNameIfTracedMap, 880
calledby mapLetPrint, 874
calls exit, 880
calls get, 880
calls seq, 880
uses /tracenames, 880
uses $InteractiveFrame, 880
defun, 880
getBrowseDatabase, 1074
calls grepConstruct, 1075
calls member, 1075
uses $includeUnexposed?, 1075
defun, 1074
getConstructorForm
calledby make-databases, 1010
getConstructorSignature
calledby evaluateType1, 907
calledby reportOpsFromLisplib, 810
getdatabase, 1001
calledby abbQuery, 532
calledby addoperations, 998
calledby describeSpad2Cmd, 525
calledby displayOperationsFromLisplib, 812
calledby displayValue, 455
calledby domainToGenvar, 851
calledby evaluateType1, 908
calledby hasAtt, 422
calledby initial-getdatabase, 992

calledby loadLibNoUpdate, 1031
calledby loadLib, 1029
calledby localnrlib, 1007
calledby mkEvalable, 903
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calledby setExposeAddConstr, 692
calledby setExposeDropConstr, 695
calledby showdatabase, 999
calledby whatConstructors, 935
calls unsqueeze, 1001
calls warn, 1001
uses *browse-stream*, 1001
uses *category-stream*, 1001
uses *defaultdomain-list*, 1001
uses *hasCategory-hash*, 1001
uses *hascategory-hash*, 1001
uses *interp-stream*, 1001
uses *miss*, 1001
uses *operation-hash*, 1001
uses *operation-stream*, 1001
uses $spadroot, 1001
defun, 1001
getDirectoryList, 974
uses $UserLevel, 974
uses $current-directory, 974
uses $directory-list, 974
uses $library-directory-list, 974
defun, 974
getEnv
calledby DaaseName, 1014
calledby initial-getdatabase, 992
calledby make-databases, 1010
calledby restart0, 19
getenv
calledby getenviron, 31
getenviron, 31
calledby copyright, 518
calledby initroot, 35
calledby summary, 822
calls getenv, 31
defun, 31
getErFromDbL
calledby getMsgInfoFromKey, 364
getFirstWord, 465
calledby doSystemCommand, 442
calls stringSpaces, 465
calls subseq, 465
defun, 465
getFlag
calledby interpFunctionDepAlists, 461
getI
calledby displayProperties, 457
calledby fetchOutput, 596
calledby getAndSay, 457
getInfovec
calledby hasAtt, 422
getInterpMacroNames
calledby clearCmdParts, 501
calledby displayMacros, 534
calledby displayProperties, 457
calledby displayWorkspaceNames, 450
getIProplist
calledby displayProperties, 457
getKeyedMsg, 327
calledby msgText, 62
calledby sayKeyedMsgLocal, 328
calls fetchKeyedMsg, 327
defun, 327
getl
calledby reportOpsFromUnitDirectly, 814
getLinePos, 370
calledby makeMsgFromLine, 369
defun, 370
getLineText, 370
calledby makeMsgFromLine, 369
defun, 370
getMapSig, 843
calledby saveMapSig, 843
calls boot-equal, 843
calls get, 843
uses $InteractiveFrame, 843
defun, 843
getMapSubNames, 859
calledby traceSpad2Cmd, 837
calls getPreviousMapSubNames, 859
calls get, 859
calls unionq, 859
calls union, 859
uses /tracenames, 859
uses $InteractiveFrame, 859
uses $lastUntraced, 859

defun, 859
getMsgArgL, 347
calledby getMsgInfoFromKey, 364
calledby sameMsg?, 374
defun, 347
getMsgCatAttr, 356
calledby getMsgToWhere, 361
calledby initToWhere, 366
calledby msgImPr?, 356
calledby msgNoRep?, 373
calls ifcdr, 356
calls ncAlist, 356
calls qassq, 356
defun, 356
getMsgFTTag
usedby posPointers, 376
getMsgFTTag?, 357
calledby getMsgPos2, 376
calledby getMsgPos, 357
calledby processChPosesForOneLine, 374
calledby putFTText, 377
calls getMsgPosTagOb, 357
calls ifcar, 357
defun, 357
getMsgInfoFromKey, 364
calledby putDatabaseStuff, 363
calls getErFromDbL, 364
calls getMsgArgL, 364
calls getMsgKey?, 364
calls getMsgKey, 364
calls removeAttributes, 364
calls segmentKeyedMsg, 364
calls substituteSegmentedMsg, 364
uses $msgDatabaseName, 364
defun, 364
getMsgKey, 346
calledby getMsgInfoFromKey, 364
calledby processKeyedError, 351
calledby sameMsg?, 374
defun, 346
getMsgKey?, 360
calledby getMsgInfoFromKey, 364
calledby getMsgLitSym, 360
calledby getStFromMsg, 352
calls identp, 360
defun, 360
getMsgLitSym, 360
calledby getStFromMsg, 352
calls getMsgKey?, 360
defun, 360
getMsgPos, 357
calledby erMsgCompare, 368
calledby erMsgSep, 368
calledby getPosStL, 354
calledby posPointers, 376
calledby processChPosesForOneLine, 374
calledby processMsgList, 367
calledby putFTText, 377
calls getMsgFTTag?, 357
calls getMsgPosTagOb, 357
defun, 357
getMsgPos2, 376
calledby posPointers, 376
calledby putFTText, 377
calls getMsgFTTag?, 376
calls getMsgPosTagOb, 376
calls ncBug, 377
defun, 376
getMsgPosTagOb, 346
calledby getMsgFTTag?, 357
calledby getMsgPos2, 376
calledby getMsgPos, 357
defun, 346
getMsgPrefix, 347
calledby processChPosesForOneLine, 374
defun, 347
getMsgPrefix?, 348
calledby getStFromMsg, 352
calledby processKeyedError, 351
calledby tabbing, 360
defun, 348
getMsgTag, 348
calledby getMsgTag?, 348
calledby getStFromMsg, 352
calledby initImPr, 365
calledby leader?, 349
calledby line?, 349
calls ncTag, 348
defun, 348
getMsgTag?, 348
calledby processKeyedError, 351
calls IFCAR, 348

calls getMsgTag, 348
defun, 348
getMsgText, 347
calledby getStFromMsg, 352
calledby putFTText, 377
defun, 347
getMsgToWhere, 361
calledby toFile?, 361
calledby toScreen?, 349
calls getMsgCatAttr, 361
defun, 361
getOperationAlistFromLisplib
calledby hasSig, 421
calledby spadTrace, 866
getOplistForConstructorForm, 816
calledby reportOpsFromUnitDirectly, 814
defun, 816
getOplistWithUniqueSignatures, 817
defun, 817
getOption, 882
calledby spadTrace, 866
calledby spadUntrace, 885
calledby traceDomainConstructor, 870
calls assoc, 882
defun, 882
getParserMacroNames, 449
calledby clearCmdParts, 501
calledby displayMacros, 534
calledby displayProperties, 457
calledby displayWorkspaceNames, 450
uses $pfMacros, 449
defun, 449
getPosStL, 354
calledby getStFromMsg, 352
calls decideHowMuch, 354
calls getMsgPos, 354
calls listDecideHowMuch, 354
calls msgImPr?, 354
calls ppos, 354
calls remFile, 354
calls remLine, 354
calls showMsgPos?, 354
uses $lastPos, 354
defun, 354
getPreStL, 353
calledby getStFromMsg, 352
calls size, 353
uses $preLength, 353
defun, 353
getPreviousMapSubNames, 860
calledby getMapSubNames, 859
calledby untraceMapSubNames, 863
calls exit, 860
calls get, 860
calls seq, 860
defun, 860
getProplist, 954
calledby addBinding, 953
calledby getProplist, 954
calls getProplist, 954
calls search, 954
uses $CategoryFrame, 954
defun, 954
getrefv32, 1048
defun, 1048
getStFromMsg, 352
calledby msgOutputter, 351
calls getMsgKey?, 352
calls getMsgLitSym, 352
calls getMsgPrefix?, 352
calls getMsgTag, 352
calls getMsgText, 352
calls getPosStL, 352
calls getPreStL, 352
calls pname, 352
calls tabbing, 352
defun, 352
getSystemCommandLine, 825
calledby synonymSpad2Cmd, 824
calls strpos, 825
calls substring, 825
uses $currentLine, 826
defun, 825
getTraceOption, 844
calledby getTraceOptions, 842
calledby trace1, 838
calls concat, 844
calls exit, 844
calls getTraceOption,hn, 844
calls identp, 844
calls isListOfIdentifiersOrStrings, 844
calls isListOfIdentifiers, 844

calls object2String, 844
calls qcar, 844
calls qcdr, 844
calls selectOptionLC, 844
calls seq, 844
calls stackTraceOptionError, 844
calls throwKeyedMsg, 844
calls transOnlyOption, 844
uses $traceOptionList, 844
defun, 844
getTraceOption,hn, 843
calledby getTraceOption, 844
calls domainToGenvar, 844
calls exit, 843
calls isDomainOrPackage, 844
calls seq, 843
calls stackTraceOptionError, 844
defun, 843
getTraceOptions, 842
calledby trace1, 838
calls exit, 842
calls getTraceOption, 842
calls poundsign, 842
calls seq, 842
calls throwKeyedMsg, 842
calls throwListOfKeyedMsgs, 842
uses $traceErrorStack, 842
defun, 842
getValue
calledby evaluateType, 906
calledby getAndEvalConstructorArgument, 948
calledby interpret1, 54
getWorkspaceNames, 451
calledby displayProperties, 457
calledby displayWorkspaceNames, 450
calls exit, 451
calls nmsort, 451
calls seq, 451
uses $InteractiveFrame, 451
defun, 451
grepConstruct
calledby getBrowseDatabase, 1075

handleNoParseCommands, 441
calledby doSystemCommand, 442
calls concat, 466
calls member, 466
calls npboot, 466
calls nplisp, 466
calls npsynonym, 466
calls npsystem, 466
calls sayKeyedMsg, 466
calls stripLisp, 466
calls stripSpaces, 466
defun, 441
handleParsedSystemCommands, 464
calledby doSystemCommand, 442
calls dumbTokenize, 464
calls parseSystemCmd, 464
calls systemCommand, 464
calls tokTran, 464
defun, 464
handleTokensizeSystemCommands, 443
calledby doSystemCommand, 442
calls dumbTokenize, 443
calls systemCommand, 443
calls tokTran, 443
defun, 443
hasAtt, 422
calledby hasAttSig, 424
calledby hasCaty, 418
calls constructSubst, 422
calls getInfovec, 422
calls getdatabase, 422
calls hasCatExpression, 422
calls subCopy, 422
calls unifyStruct, 422
local def $domPvar, 422
defun, 422
hasAttSig, 424
calledby hasCaty, 418
calls hasAtt, 424
calls hasSig, 424
calls keyedSystemError, 424
defun, 424
hasCat
calledby hasCaty, 418
hasCate, 431
calledby hasCatExpression, 425
calledby hasCate1, 425
calledby hasCateSpecial, 432

calledby hasCaty1, 429
calledby hasSigAnd, 423
calledby hasSigOr, 424
calledby hasSig, 421
calls containsVariables, 431
calls hasCate1, 431
calls hasCateSpecial, 431
calls hasCaty, 431
calls isPatternVar, 431
calls subCopy, 431
local def $hope, 431
local ref $EmptyMode, 431
local ref $Subst, 431
defun, 431
hasCate1, 425
calledby hasCate, 431
calls hasCate, 425
local def $domPvar, 425
defun, 425
HasCategory
calledby basicLookup, 1061
hasCateSpecial, 432
calledby hasCate, 431
calls augmentSub, 432
calls canCoerceFrom, 432
calls containsVars, 432
calls eqcar, 432
calls hasCateSpecialNew, 432
calls hasCate, 432
calls hasCaty, 432
calls isSubDomain, 432
local ref $Integer, 432
local ref $QuotientField, 432
defun, 432
hasCateSpecialNew, 433
calledby hasCateSpecial, 432
calls augmentSub, 433
calls defaultTargetFE, 433
calls eqcar, 433
calls hasCaty, 433
calls isEqualOrSubDomain, 433
calls member, 433
calls underDomainOf, 433
local ref $ComplexInteger, 433
local ref $Integer, 433
local ref $RationalNumber, 433
defun, 433
hasCatExpression, 425
calledby hasAtt, 422
calledby hasCatExpression, 425
calls hasCatExpression, 425
calls hasCate, 425
calls keyedSystemError, 425
defun, 425
hasCaty, 418
calledby hasCateSpecialNew, 433
calledby hasCateSpecial, 432
calledby hasCate, 431
calledby ofCategory, 417
calls augmentSub, 418
calls constructSubst, 418
calls domArg2, 418
calls domArg, 418
calls hasAttSig, 418
calls hasAtt, 418
calls hasCaty1, 419
calls hasCat, 418
calls hasSig, 418
calls kdr, 418
calls mkDomPvar, 418
calls opOf, 418
calls subCopy, 418
calls unifyStruct, 418
local ref $domPvar, 419
defun, 418
hasCaty1, 429
calledby hasCaty1, 429
calledby hasCaty, 419
calls hasCate, 429
calls hasCaty1, 429
calls keyedSystemError, 429
local def $domPvar, 429
defun, 429
hashable, 1060
calls Boolean, 1060
calls bpiname, 1060
calls compiledLookup, 1060
calls knownEqualPred, 1060
defun, 1060
hashCode?
calledby basicLookupCheckDefaults, 1063
calledby basicLookup, 1061

hashString
calledby basicLookupCheckDefaults, 1063
calledby basicLookup, 1061
hashtable-class
calledby writify,writifyInner, 603
hashType
calledby basicLookupCheckDefaults, 1063
calledby basicLookup, 1061
hasOptArgs?, 307
calledby pfApplication2Sex, 304
defun, 307
hasOption, 447
calledby trace1, 838
calls pname, 447
calls stringPrefix?, 447
defun, 447
hasPair, 881
calledby hasPair, 881
calledby letPrint2, 877
calledby letPrint3, 878
calledby letPrint, 875
calls hasPair, 881
calls qcar, 881
calls qcdr, 881
defun, 881
hasSig, 421
calledby hasAttSig, 424
calledby hasCaty, 418
calls assq, 421
calls cnstructSubst, 421
calls constructor?, 421
calls getOperationAlistFromLisplib, 421
calls hasCate, 421
calls hasSigAnd, 421
calls hasSigOr, 421
calls keyedSystemError, 421
calls subCopy, 421
calls unifyStruct, 421
local def $domPvar, 421
defun, 421
hasSigAnd, 423
calledby hasSigOr, 424
calledby hasSig, 421
calls hasCate, 423
calls keyedSystemError, 423
calls subCopy, 423
defun, 423
hasSigOr, 424
calledby hasSig, 421
calls hasCate, 424
calls hasSigAnd, 424
calls keyedSystemError, 424
defun, 424
help, 568
calls helpSpad2Cmd, 568
defun, 568
help help page, 565
manpage, 565
helpSpad2Cmd, 568
calledby abbreviationsSpad2Cmd, 479
calledby frameSpad2Cmd, 562
calledby help, 568
calledby savesystem, 642
calledby showSpad2Cmd, 806
calledby systemCommand, 444
calls newHelpSpad2Cmd, 568
calls sayKeyedMsg, 568
defun, 568
hget, 1038
calledby ScanOrPairVec,ScanOrInner, 611
calledby dewritify,dewritifyInner, 608
calledby keyword?, 118
calledby keyword, 118
calledby saveDependentsHashTable, 1013
calledby saveUsersHashTable, 1014
calledby writify,writifyInner, 603
defmacro, 1038
histFileErase, 613
calledby disableHist, 599
calledby historySpad2Cmd, 578
calledby initHist, 577
calledby restoreHistory, 593
calledby saveHistory, 591
calledby setHistoryCore, 581
calledby writeInputLines, 583
defun, 613
histFileName, 576
calledby addNewInterpreterFrame, 557
calledby clearCmdAll, 499
calledby disableHist, 599
calledby historySpad2Cmd, 578
calledby initHist, 577

calledby readHiFi, 597
calledby restoreHistory, 593
calledby saveHistory, 591
calledby setHistoryCore, 581
calledby writeHiFi, 598
calls makeHistFileName, 576
uses $interpreterFrameName, 576
defun, 576
histInputFileName, 576
calledby saveHistory, 591
calledby writeInputLines, 583
calls makePathname, 576
uses $historyDirectory, 576
uses $interpreterFrameName, 576
defun, 576
history, 578
calls historySpad2Cmd, 578
calls sayKeyedMsg, 578
uses $options, 578
defun, 578
history help page, 571
manpage, 571
historySpad2Cmd, 578
calledby history, 578
calls changeHistListLen, 578
calls clearSpad2Cmd, 578
calls disableHist, 578
calls exit, 579
calls histFileErase, 578
calls histFileName, 578
calls initHistList, 578
calls member, 578
calls queryUserKeyedMsg, 578
calls resetInCoreHist, 578
calls restoreHistory, 578
calls saveHistory, 578
calls sayKeyedMsg, 578
calls selectOptionLC, 578
calls seq, 579
calls setHistoryCore, 578
calls showHistory, 578
calls string2id-n, 578
calls upcase, 578
calls writeInputLines, 578
uses $HiFiAccess, 579
uses $IOindex, 579
uses $options, 579
defun, 578
hkeys, 1038
calledby saveDependentsHashTable, 1013
calledby saveUsersHashTable, 1014
calledby scanDictCons, 135
calledby scanPunCons, 137
calledby writify,writifyInner, 603
defun, 1038
hput, 1038
calledby ScanOrPairVec,ScanOrInner, 611
calledby addBinding, 953
calledby dewritify,dewritifyInner, 608
calledby writify,writifyInner, 603
defun, 1038

id
calledby inclmsgConActive, 93
calledby inclmsgConStill, 94
calledby inclmsgFileCycle, 92
calledby inclmsgIfSyntax, 97
calledby inclmsgSay, 90
idChar?, 125
calledby scanW, 125
defmacro, 125
identp, 1040
calledby functionp, 1041
calledby getMsgKey?, 360
calledby getTraceOption, 844
calledby isDomainValuedVariable, 949
calledby isListOfIdentifiersOrStrings, 859
calledby isListOfIdentifiers, 858
calledby isSharpVar, 876
calledby isgenvar, 877
calledby messageprint-1, 1043
calledby ncAlist, 413
calledby ncTag, 413
calledby ofCategory, 417
calledby restoreHistory, 593
calledby rwrite, 601
calledby selectOption, 475
calledby undo, 912
defmacro, 1040
if, 393
syntax, 393
If?, 79

calledby incLude1, 82
calls quotient, 79
defvar, 79
IFCAR
calledby getMsgTag?, 348
calledby posPointers, 376
calledby remLine, 355
ifcar
calledby defaultTargetFE, 435
calledby fillerSpaces, 20
calledby getMsgFTTag?, 357
calledby pfAbSynOp, 410
calledby pfExpression, 262
calledby pfLeaf, 245
calledby pfSymbol, 249
calledby pfSymb, 249
calledby tokConstruct, 409
IFCDR
calledby remFile, 355
ifcdr
calledby clearParserMacro, 449
calledby getMsgCatAttr, 356
calledby mac0Get, 226
calledby setMsgCatlessAttr, 363
calledby specialChar, 970
ifCond, 89
calledby incLude1, 82
calls MakeSymbol, 89
calls incCommandTail, 89
uses $inclAssertions, 89
defun, 89
IfKeepPart, 77
defvar, 77
IfSkipPart, 78
defvar, 78
IfSkipToEnd, 77
defvar, 77
images
Restart, 16
import
calledby frameSpad2Cmd, 562
importFromFrame, 559
calledby frameSpad2Cmd, 562
calledby importFromFrame, 559
calls boot-equal, 559
calls clearCmdParts, 559
calls exit, 559
calls frameEnvironment, 559
calls frameNames, 559
calls framename, 559
calls getalist, 559
calls get, 559
calls importFromFrame, 559
calls member, 559
calls putHist, 559
calls queryUserKeyedMsg, 559
calls sayKeyedMsg, 559
calls seq, 559
calls string2id-n, 559
calls throwKeyedMsg, 559
calls upcase, 559
uses $interpreterFrameRing, 559
defun, 559
in-stream, 951
usedby ncTopLevel, 25
usedby serverReadLine, 45
defvar, 951
incActive?, 103
calledby incLude1, 81
defun, 103
incAppend, 87
calledby incAppend1, 87
calledby incLude1, 82
calledby next1, 39
calls Delay, 87
calls incAppend1, 87
defun, 87
incAppend1, 87
calledby incAppend, 87
calls StreamNull, 87
calls incAppend, 87
defun, 87
incBiteOff, 619
calledby incFileName, 618
defun, 619
incClassify, 99
calledby incLude1, 82
calls incCommand?, 99
uses incCommands, 99
defun, 99
incCommand?, 100
calledby incClassify, 99

calls char, 100
defun, 100
incCommands, 98
usedby incClassify, 99
defvar, 98
incCommandTail, 101
calledby assertCond, 96
calledby ifCond, 89
calledby incLude1, 81
calledby inclFname, 102
calls incDrop, 101
defun, 101
incConsoleInput, 102
calledby incLude1, 82
calls incRgen, 102
calls make-instream, 102
defun, 102
incDrop, 101
calledby incCommandTail, 101
calls substring, 101
defun, 101
incFileInput, 102
calledby incLude1, 82
calls incRgen, 102
calls make-instream, 102
defun, 102
incFileName, 618
calledby inclFname, 102
calledby ncloopIncFileName, 618
calls incBiteOff, 618
defun, 618
incHandleMessage, 76
calledby incRenumberLine, 75
calls ncBug, 76
calls ncSoftError, 76
defun, 76
incIgen, 75
calledby incIgen1, 75
calledby incRenumber, 74
calls Delay, 75
calls incIgen1, 75
defun, 75
incIgen1, 75
calledby incIgen, 75
calls incIgen, 75
defun, 75
inclFname, 102
calledby incLude1, 82
calls incCommandTail, 102
calls incFileName, 102
defun, 102
incLine, 87
calledby xlMsg, 86
calledby xlSkip, 89
calls incLine1, 87
defun, 87
incLine1, 88
calledby incLine, 87
calledby xlOK1, 86
calls lnCreate, 88
defun, 88
inclmsgCannotRead, 92
calledby xlCannotRead, 91
calls thefname, 92
defun, 92
inclmsgCmdBug, 98
calledby xlCmdBug, 98
defun, 98
inclmsgConActive, 93
calledby xlConActive, 93
calls id, 93
defun, 93
inclmsgConsole, 94
calledby xlConsole, 94
defun, 94
inclmsgConStill, 94
calledby xlConStill, 94
calls id, 94
defun, 94
inclmsgFileCycle, 92
calledby xlFileCycle, 92
calls id, 92
calls porigin, 92
defun, 92
inclmsgFinSkipped, 95
calledby xlSkippingFin, 95
defun, 95
inclmsgIfBug, 97
calledby xlIfBug, 97
defun, 97
inclmsgIfSyntax, 97
calledby xlIfSyntax, 96

calls concat, 97
calls id, 97
calls origin, 97
defun, 97
inclmsgNoSuchFile, 91
calledby xlNoSuchFile, 90
calls thefname, 91
defun, 91
inclmsgPrematureEOF, 88
calledby xlPrematureEOF, 86
calls origin, 88
defun, 88
inclmsgPrematureFin, 95
calledby xlPrematureFin, 95
calls origin, 95
defun, 95
inclmsgSay, 90
calledby xlSay, 90
calls id, 90
defun, 90
incLude, 76
calledby incLude1, 81
calledby incStream, 73
calledby incString, 39
calls Delay, 76
defun, 76
include
calls incLude1, 76
include help page, 617
manpage, 617
incLude1, 81
calledby include, 76
calls Elseif?, 82
calls If?, 82
calls KeepPart?, 82
calls Rest, 81
calls SkipEnd?, 82
calls SkipPart?, 82
calls Skipping?, 81
calls StreamNull, 81
calls Top?, 81
calls assertCond, 82
calls concat, 81
calls expand-tabs, 82
calls ifCond, 82
calls incActive?, 81
calls incAppend, 82
calls incClassify, 82
calls incCommandTail, 81
calls incConsoleInput, 82
calls incFileInput, 82
calls incLude, 81
calls incNConsoles, 82
calls inclFname, 82
calls xlCannotRead, 81
calls xlCmdBug, 82
calls xlConActive, 82
calls xlConStill, 82
calls xlConsole, 82
calls xlFileCycle, 81
calls xlIfBug, 82
calls xlIfSyntax, 82
calls xlNoSuchFile, 81
calls xlOK1, 81
calls xlOK, 81
calls xlPrematureEOF, 81
calls xlPrematureFin, 82
calls xlSay, 81
calls xlSkippingFin, 82
calls xlSkip, 81
defun, 81
incNConsoles, 103
calledby incLude1, 82
calledby incNConsoles, 103
calls incNConsoles, 103
defun, 103
incPrefix?, 100
calledby lineoftoks, 109
calledby scanIgnoreLine, 110
defun, 100
incRenumber, 74
calledby incStream, 73
calledby incString, 39
calls incIgen, 74
calls incZip, 74
defun, 74
incRenumberItem, 76
calledby incRenumberLine, 75
calls lnSetGlobalNum, 76
defun, 76
incRenumberLine, 75
calls incHandleMessage, 75

calls incRenumberItem, 75
defun, 75
incRgen, 103
calledby incConsoleInput, 102
calledby incFileInput, 102
calledby incRgen1, 104
calledby incStream, 73
calls Delay, 103
calls incRgen1, 103
defun, 103
incRgen1, 104
calledby incRgen, 103
calls incRgen, 104
uses StreamNil, 104
defun, 104
incStream, 73
calledby intloopInclude0, 63
calledby ncloopInclude0, 73
calls incLude, 73
calls incRenumber, 73
calls incRgen, 73
uses Top, 73
defun, 73
incString, 39
calledby intloopProcessString, 37
calledby parseFromString, 48
calls incLude, 39
calls incRenumber, 39
uses Top, 39
defun, 39
incZip, 74
calledby incRenumber, 74
calledby incZip1, 74
calls Delay, 74
calls incZip1, 74
defun, 74
incZip1, 74
calledby incZip, 74
calls StreamNull, 74
calls incZip, 74
defun, 74
infgeneric, 107
defvar, 107
init-boot/spad-reader, 958
calls ioclear, 958
calls next-lines-clear, 958
uses *standard-output* , 958
uses $spad-errors, 958
uses boot-line-stack, 958
uses file-closed, 958
uses line-handler, 958
uses meta-error-handler, 958
uses spaderrorstream, 958
uses xtokenreader, 958
defun, 958
init-memory-config, 34
calledby restart, 17
calls allocate-contiguous-pages, 34
calls allocate-relocatable-pages, 34
calls allocate, 34
calls set-hole-size, 34
defun, 34
initHist, 577
calledby restart, 17
calls $replace, 577
calls histFileErase, 577
calls histFileName, 577
calls initHistList, 577
calls makeInputFilename, 577
calls oldHistFileName, 577
uses $HiFiAccess, 577
uses $useInternalHistoryTable, 577
defun, 577
initHistList, 577
calledby addNewInterpreterFrame, 557
calledby historySpad2Cmd, 578
calledby initHist, 577
uses $HistListAct, 577
uses $HistListLen, 577
uses $HistList, 577
uses $HistRecord, 577
defun, 577
initial-getdatabase, 992
calledby resethashtables, 991
calls getEnv, 992
calls getdatabase, 992
defun, 992
initializeInterpreterFrameRing, 551
calledby restart, 17
calls emptyInterpreterFrame, 551
calls updateFromCurrentInterpreterFrame, 551

uses $interpreterFrameName, 551
uses $interpreterFrameRing, 551
defun, 551
initializeSetVariables, 645
calledby initializeSetVariables, 645
calledby resetWorkspaceVariables, 646
calls initializeSetVariables, 645
calls literals, 645
calls sayMSG, 645
calls translateYesNo2TrueFalse, 645
calls tree, 645
defun, 645
initializeTimedNames
calledby processInteractive, 49
initImPr, 365
calledby msgCreate, 346
calls getMsgTag, 365
calls setMsgUnforcedAttr, 365
uses $erMsgToss, 365
uses $imPrTagGuys, 365
defun, 365
initroot, 35
calledby restart, 17
calls getenviron, 35
calls reroot, 35
uses $spadroot, 35
defun, 35
initToWhere, 366
calledby msgCreate, 346
calls getMsgCatAttr, 366
calls setMsgUnforcedAttr, 366
defun, 366
input-libraries, 660
usedby addInputLibrary, 660
usedby dropInputLibrary, 661
usedby openOutputLibrary, 658
usedby setInputLibrary, 659
defvar, 660
insert
calledby updateSourceFiles, 542
insertpile, 333
calledby intloopInclude0, 63
calledby ncloopInclude0, 73
calls npNull, 333
calls pileCforest, 333
calls pilePlusComments, 333
calls pilePlusComment, 333
calls pileTree, 333
defun, 333
insertPos, 377
calledby posPointers, 376
calls done, 377
defun, 377
installConstructor
calledby loadLibNoUpdate, 1031
calledby loadLib, 1029
calledby localnrlib, 1007
integer-decode-float-denominator, 1088
defun, 1088
integer-decode-float-exponent, 1088
defun, 1088
integer-decode-float-numerator, 1087
defun, 1087
integer-decode-float-sign, 1088
defun, 1088
internl
calledby constructSubst, 432
calledby spadTraceAlias, 881
InterpExecuteSpadSystemCommand, 32
calls ExecuteInterpSystemCommand, 32
uses $intCoerceFailure, 32
uses $intSpadReader, 32
catches, 32
defun, 32
interpFunctionDepAlists, 461
calledby displayProperties, 457
calls getFlag, 461
calls getalist, 461
calls putalist, 461
uses $InteractiveFrame, 461
uses $dependeeAlist, 461
uses $dependentAlist, 461
uses $e, 461
defun, 461
interpOpen, 994
calls DaaseName, 995
calls make-database, 995
calls unsqueeze, 995
uses *allconstructors*, 995
uses *interp-stream*, 995
uses *interp-stream-stamp*, 995
uses $spadroot, 995

defun, 994
interpopen
calledby resethashtables, 991
calledby restart0, 19
interpret, 54
calledby interpretTopLevel, 53
calls interpret1, 54
uses $env, 54
uses $eval, 54
uses $genValue, 54
defun, 54
interpret1, 54
calledby interpret, 54
calls bottomUp, 54
calls getArgValue, 54
calls getValue, 54
calls interpret2, 54
calls keyedSystemError, 54
calls mkAtreeWithSrcPos, 54
calls objNew, 54
calls putTarget, 54
uses $eval, 54
uses $genValue, 54
defun, 54
interpret2, 55
calledby interpret1, 54
calls coerceInteractive, 55
calls member, 55
calls objMode, 55
calls objNew, 55
calls objVal, 55
calls systemErrorHere, 55
calls throwKeyedMsgCannotCoerceWithValue, 55
uses $EmptyMode, 55
uses $ThrowAwayMode, 55
defun, 55
interpretTopLevel, 53
calledby interpretTopLevel, 53
calledby processInteractive1, 52
calls interpretTopLevel, 53
calls interpret, 53
calls peekTimedName, 53
calls stopTimingProcess, 53
uses $timedNameStack, 53
catches, 53
defun, 53
interrupt
calledby break, 896
intInterpretPform, 67
calledby phInterpret, 67
calls pf2Sex, 67
calls processInteractive, 67
calls zeroOneTran, 67
defun, 67
intloop, 26
calledby ncIntLoop, 25
calls SpadInterpretStream, 26
calls resetStackLimits, 26
uses $intRestart, 26
uses $intTopLevel, 26
catches, 26
defun, 26
intloopEchoParse, 69
calledby intloopInclude0, 63
calls dqToList, 69
calls mkLineList, 69
calls ncloopDQlines, 69
calls ncloopPrintLines, 69
calls npParse, 69
calls setCurrentLine, 69
uses $EchoLines, 69
uses $lines, 69
defun, 69
intloopInclude, 63
calledby SpadInterpretStream, 29
calls ST, 63
calls intloopInclude0, 63
defun, 63
intloopInclude0, 63
calledby intloopInclude, 63
calls incStream, 63
calls insertpile, 63
calls intloopEchoParse, 63
calls intloopProcess, 63
calls lineoftoks, 63
calls next, 63
uses $lines, 63
defun, 63
intloopPrefix?, 36
calledby intloopReadConsole, 30
defun, 36

intloopProcess, 64
calledby intloopInclude0, 63
calledby intloopProcessString, 37
calledby intloopProcess, 64
calls StreamNull, 64
calls $systemCommandFunction, 64
calls intloopProcess, 64
calls intloopSpadProcess, 64
calls pfAbSynOp?, 64
calls setCurrentLine, 64
calls tokPart, 64
uses $systemCommandFunction, 64
defun, 64
intloopProcessString, 37
calledby intloopReadConsole, 30
calls incString, 37
calls intloopProcess, 37
calls next, 37
calls setCurrentLine, 37
defun, 37
intloopReadConsole, 30
calledby SpadInterpretStream, 29
calledby intloopReadConsole, 30
calls concat, 30
calls intloopPrefix?, 30
calls intloopProcessString, 30
calls intloopReadConsole, 30
calls intnplisp, 30
calls leaveScratchpad, 30
calls mkprompt, 30
calls ncloopCommand, 30
calls ncloopEscaped, 30
calls serverReadLine, 30
calls setCurrentLine, 30
uses $dalymode, 31
defun, 30
throws, 30
intloopSpadProcess, 64
calledby intloopProcess, 64
calls CatchAsCan, 65
calls Catch, 65
calls intloopSpadProcess,interp, 65
calls ncPutQ, 65
uses $NeedToSignalSessionManager, 65
uses $currentCarrier, 65
uses $intCoerceFailure, 65
uses $intSpadReader, 65
uses $ncMsgList, 65
uses $prevCarrier, 65
uses $stepNo, 65
uses flung, 65
catches, 64, 65
defun, 64
intloopSpadProcess,interp, 65
calledby intloopSpadProcess, 65
calls ncConversationPhase, 65
calls ncEltQ, 66
calls ncError, 66
defun, 65
intnplisp, 36
calledby intloopReadConsole, 30
calls nplisp, 36
uses $currentLine, 36
defun, 36
intp
calledby dewritify,dewritifyInner, 608
intProcessSynonyms, 33
calledby ExecuteInterpSystemCommand, 33
calls processSynonyms, 33
uses line, 33
defun, 33
ioclear, 962
calledby init-boot/spad-reader, 958
calledby spad-syntax-error, 959
calls line-clear, 962
calls reduce-stack-clear, 962
uses $boot, 962
uses $spad, 962
uses current-fragment, 962
uses current-line, 962
defun, 962
iostat, 960
calledby spad-long-error, 959
calls line-past-end-p, 960
calls line-print, 960
calls next-lines-show, 960
calls token-stack-show, 960
uses $boot, 960
uses $current-line, 960
uses $spad, 960
defun, 960

is-console[9]
called by serverReadLine, 45
called by shut, 972
isDomain
calledby ?t, 893
calledby addTraceItem, 892
calledby compiledLookup, 1061
isDomainOrPackage, 865
calledby /tracereply, 884
calledby ?t, 893
calledby addTraceItem, 892
calledby getTraceOption,hn, 844
calledby prTraceNames,fn, 888
calledby shortenForPrinting, 881
calledby spadReply,printName, 884
calledby spadTrace, 866
calledby spadUntrace, 885
calledby traceReply, 889
calledby untraceDomainConstructor,keepTraced2, 872
calledby writify,writifyInner, 603
calls isFunctor, 865
calls opOf, 865
calls poundsign, 865
calls refvecp, 865
defun, 865
isDomainValuedVariable, 949
calledby evaluateType, 906
calledby reportOperations, 807
calls get, 949
calls identp, 949
calls member, 949
calls objMode, 949
calls objValUnwrap, 949
local ref $InteractiveFrame, 949
local ref $env, 949
local ref $e, 949
defun, 949
isEqualOrSubDomain, 436
calledby defaultTargetFE, 435
calledby hasCateSpecialNew, 433
calls isSubDomain, 436
defun, 436
isExposedConstructor, 812
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calls getalist, 812
local ref $globalExposureGroupAlist, 812
local ref $localExposureData, 812
defun, 812
isFunctor
calledby funfind,LAM, 864
calledby isDomainOrPackage, 865
calledby trace1, 838
calledby traceReply, 889
isgenvar, 876
calledby ?t, 892
calledby letPrint2, 877
calledby letPrint3, 878
calledby letPrint, 875
calledby traceReply, 889
calledby tracelet, 894
calls digitp, 877
calls identp, 877
calls size, 876
defun, 876
isIntegerString, 464
calledby tokTran, 463
defun, 464
isInternalMapName
calledby displayProperties, 457
isInterpMacro
calledby displayMacro, 449
calledby displayProperties, 458
isInterpOnlyMap, 862
calls get, 862
uses $InteractiveFrame, 862
defun, 862
isListOfIdentifiers, 858
calledby getTraceOption, 844
calls exit, 858
calls identp, 858
calls seq, 858
defun, 858
isListOfIdentifiersOrStrings, 859
calledby getTraceOption, 844
calls exit, 859
calls identp, 859
calls seq, 859
defun, 859
isLocalVar

calledby getAndEvalConstructorArgument, 948
isMap
calledby clearCmdParts, 501
isNameOfType
calledby reportOperations, 807
isNewWorldDomain
calledby basicLookup, 1061
isPartialMode, 418
calledby evalCategory, 949
calls contained, 418
local ref $EmptyMode, 418
defun, 418
isPatternVar
calledby containsVars1, 429
calledby containsVars, 428
calledby hasCate, 431
calledby unifyStructVar, 427
calledby unifyStruct, 426
isSharpVar, 876
calledby domArg2, 420
calledby isSharpVarWithNum, 876
calls identp, 876
defun, 876
isSharpVarWithNum, 876
calledby getAliasIfTracedMapParameter, 879
calledby letPrint2, 877
calledby letPrint3, 878
calledby letPrint, 875
calls dig2fix, 876
calls digitp, 876
calls isSharpVar, 876
calls pname, 876
calls qcsize, 876
defun, 876
isSubDomain
calledby hasCateSpecial, 432
calledby isEqualOrSubDomain, 436
isSubForRedundantMapName, 863
calledby traceReply, 889
calls assocleft, 863
calls member, 863
calls rassocSub, 863
uses $mapSubNameAlist, 863
defun, 863
isSystemDirectory, 1030
calledby loadLib, 1029
calls function, 1030
local ref $spadroot, 1030
defun, 1030
isTraceGensym, 865
calls gensymp, 865
defun, 865
isType
calledby reportOperations, 808
isUncompiledMap, 861
calls get, 861
uses $InteractiveFrame, 861
defun, 861
isWrapped
calledby getAndEvalConstructorArgument, 948
calledby retract, 1049
iterate, 395
syntax, 395

justifyMyType, 62
calledby printTypeAndTimeNormal, 59
calls fillerSpaces, 62
uses $linelength, 62
defun, 62

kaddr
calledby zsystemdevelopment1, 944
kadr
calledby zsystemdevelopment1, 944
kar
calledby recordFrame, 913
calledby untraceDomainConstructor,keepTraced?, 872
calledby zsystemdevelopment1, 944
kdr
calledby hasCaty, 418
calledby reportOpsFromLisplib, 810
calledby searchCurrentEnv, 954
calledby searchTailEnv, 955
calledby set1, 800
calledby spadTrace, 866
KeepPart?, 80
calledby Skipping?, 81
calledby incLude1, 82

calls remainder, 80
defvar, 80
keyedSystemError
calledby compiledLookupCheck, 497
calledby hasAttSig, 424
calledby hasCatExpression, 425
calledby hasCaty1, 429
calledby hasSigAnd, 423
calledby hasSigOr, 424
calledby hasSig, 421
calledby interpret1, 54
calledby pf2Sex1, 299
calledby pfLiteral2Sex, 302
calledby readHiFi, 597
keyword, 118
calledby lfkey, 119
calledby scanCloser?, 123
calledby scanKeyTr, 117
calls hget, 118
defun, 118
keyword?, 118
calledby scanWord, 123
calls hget, 118
defun, 118
knownEqualPred
calledby hashable, 1060

lassoc
calledby breaklet, 895
calledby coerceTraceFunValue2E, 857
calledby genDomainTraceName, 852
calledby letPrint2, 877
calledby letPrint3, 878
calledby letPrint, 875
calledby processSynonyms, 33
calledby reportUndo, 918
calledby serverReadLine, 44
calledby set1, 800
calledby subTypes, 856
calledby trace1, 838
calledby tracelet, 894
calledby undoSingleStep, 921
calledby unifyStructVar, 427
lassocSub, 861
calledby untrace, 852
calls lassq, 861
defun, 861
lassq
calledby diffAlist, 915
calledby lassocSub, 861
last
calledby frameSpad2Cmd, 562
lastc
calledby StringToDir, 1076
lastTokPosn, 339
calledby enPile, 338
calledby separatePiles, 339
calls tokPosn, 339
defun, 339
leader?, 349
calledby msgOutputter, 351
calledby showMsgPos?, 356
calls getMsgTag, 349
defun, 349
leave, 396
syntax, 396
leaveScratchpad, 635
calledby intloopReadConsole, 30
calledby quitSpad2Cmd, 634
defun, 635
length
calledby mkDomPvar, 430
letPrint, 875
calledby mapLetPrint, 874
calls break, 875
calls bright, 875
calls gensymp, 875
calls hasPair, 875
calls isSharpVarWithNum, 875
calls isgenvar, 875
calls lassoc, 875
calls pname, 875
calls sayBrightlyNT, 875
calls shortenForPrinting, 875
uses $letAssoc, 875
defun, 875
letPrint2, 877
calls break, 877
calls bright, 877
calls gensymp, 877
calls hasPair, 877
calls isSharpVarWithNum, 877

calls isgenvar, 877
calls lassoc, 877
calls mathprint, 877
calls pname, 877
calls print, 877
uses $BreakMode, 877
uses $letAssoc, 877
catches, 877
defun, 877
letPrint3, 878
calls break, 878
calls bright, 878
calls gensymp, 878
calls hasPair, 878
calls isSharpVarWithNum, 878
calls isgenvar, 878
calls lassoc, 878
calls mathprint, 878
calls pname, 878
calls print, 878
calls spadcall, 878
uses $BreakMode, 878
uses $letAssoc, 878
catches, 878
defun, 878
lfcomment, 114
calledby scanComment, 113
defun, 114
lferror, 133
calledby scanError, 133
defun, 133
lffloat, 125
calledby scanExponent, 124
calls concat, 125
defun, 125
lfid, 112
calledby scanToken, 111
calledby scanWord, 123
defun, 112
lfinteger, 131
calledby scanNumber, 129
defun, 131
lfkey, 119
calledby scanKeyTr, 117
calledby scanPossFloat, 118
calledby scanWord, 123
calls keyword, 119
defun, 119
lfnegcomment, 115
calledby scanNegComment, 114
defun, 115
lfrinteger, 131
calledby scanNumber, 129
calls concat, 131
defun, 131
lfspaces, 127
calledby scanSpace, 127
defun, 127
lfstring, 128
calledby scanString, 127
defun, 128
library, 1005
calledby with, 937
calls extendLocalLibdb, 1005
calls localdatabase, 1005
calls tersyscommand, 1005
uses $newConlist, 1005
uses $options, 1005
defun, 1005
library help page, 621
manpage, 621
libstream, 1148
defstruct, 1148
line
usedby commandErrorIfAmbiguous, 466
usedby commandErrorMessage, 446
usedby doSystemCommand, 442
usedby intProcessSynonyms, 33
usedby processSynonyms, 34
usedby unAbbreviateKeyword, 466
line-clear
calledby ioclear, 962
line-handler, 957
usedby init-boot/spad-reader, 958
defvar, 957
line-past-end-p
calledby iostat, 960
calledby spad-short-error, 960
line-print
calledby iostat, 960
calledby spad-short-error, 960
line?, 349

calledby msgOutputter, 351
calls getMsgTag, 349
defun, 349
lineoftoks, 108
calledby intloopInclude0, 63
calledby ncloopInclude0, 73
calledby parseFromString, 48
calls constoken, 109
calls dqUnit, 109
calls incPrefix?, 109
calls nextline, 109
calls scanIgnoreLine, 109
calls substring, 109
uses $floatok, 109
uses $f, 109
uses $linepos, 109
uses $ln, 109
uses $n, 109
uses $r, 109
uses $sz, 109
defun, 108
lisp help page, 623
manpage, 623
listConstructorAbbreviations, 480
calledby abbreviationsSpad2Cmd, 479
calledby displaySpad2Cmd, 531
calls queryUserKeyedMsg, 480
calls sayKeyedMsg, 481
calls string2id-n, 480
calls upcase, 480
calls whatSpad2Cmd, 481
defun, 480
listDecideHowMuch, 359
calledby getPosStL, 354
calls poGlobalLinePosn, 359
calls poNopos?, 359
calls poPosImmediate?, 359
defun, 359
ListMember?
calledby whichCat, 363
listOutputter, 352
calledby processMsgList, 367
calls msgOutputter, 352
defun, 352
listSort
calledby erMsgSort, 367
literals
calledby displaySetOptionInformation, 647
calledby displaySetVariableSettings, 649
calledby initializeSetVariables, 645
calledby set1, 801
lnCreate, 343
calledby incLine1, 88
defun, 343
lnExtraBlanks, 343
calledby scanError, 132
calledby scanS, 128
calledby scanToken, 112
defun, 343
lnFileName, 345
calledby poFileName, 358
calls lnFileName?, 345
calls ncBug, 345
defun, 345
lnFileName?, 345
calledby lnFileName, 345
calledby lnImmediate?, 345
defun, 345
lnGlobalNum, 344
calledby poGlobalLinePosn, 72
defun, 344
lnImmediate?, 345
calledby poPosImmediate?, 358
calls lnFileName?, 345
defun, 345
lnLocalNum, 344
calledby poLinePosn, 359
defun, 344
lnPlaceOfOrigin, 344
defun, 344
lnSetGlobalNum, 344
calledby incRenumberItem, 76
defun, 344
lnString, 343
defun, 343
load, 625
calls sayKeyedMsg, 625
defun, 625
load help page, 625
manpage, 625
loadFunctor, 1032
calledby loadFunctor, 1032

calledby traceDomainConstructor, 871
calls loadFunctor, 1032
calls loadLibIfNotLoaded, 1032
defun, 1032
loadIfNecessary
calledby mkEvalable, 903
loadLib, 1029
calledby browse, 489
calls categoryForm?, 1029
calls clearConstructorCache, 1029
calls getdatabase, 1029
calls installConstructor, 1029
calls isSystemDirectory, 1029
calls loadLibNoUpdate, 1029
calls namestring, 1029
calls pathnameDirectory, 1029
calls remprop, 1030
calls sayKeyedMsg, 1029
calls startTimingProcess, 1029
calls stopTimingProcess, 1030
calls updateCategoryTable, 1029
calls updateDatabase, 1029
local def $CategoryFrame, 1030
local ref $InteractiveMode, 1030
local ref $forceDatabaseUpdate, 1030
local ref $printLoadMsgs, 1030
defun, 1029
loadLibIfNotLoaded
calledby loadFunctor, 1032
loadLibNoUpdate, 1031
calledby loadLib, 1029
calledby localnrlib, 1007
calls clearConstructorCache, 1031
calls getdatabase, 1031
calls installConstructor, 1031
calls sayKeyedMsg, 1031
calls stopTimingProcess, 1031
calls toplevel, 1031
local def $CategoryFrame, 1031
local ref $InteractiveMode, 1031
local ref $printLoadMsgs, 1031
defun, 1031
localdatabase, 1005
calledby library, 1005
calledby make-databases, 1010
calls localnrlib, 1005
calls sayKeyedMsg, 1005
uses *index-filename*, 1005
uses $ConstructorCache, 1005
uses $forceDatabaseUpdate, 1005
defun, 1005
localnrlib, 1007
calledby localdatabase, 1005
calls addoperations, 1007
calls categoryForm?, 1007
calls getdatabase, 1007
calls installConstructor, 1007
calls loadLibNoUpdate, 1007
calls make-database, 1007
calls sayKeyedMsg, 1007
calls setExposeAddConstr, 1007
calls startTimingProcess, 1007
calls sublislis, 1007
calls updateCategoryTable, 1007
calls updateDatabase, 1007
uses *allOperations*, 1007
uses *allconstructors*, 1007
uses $FormalMapVariableList, 1007
defun, 1007
lookupInDomainVector, 1063
calledby basicLookup, 1061
calledby oldCompLookup, 1064
calls basicLookupCheckDefaults, 1063
calls spadcall, 1063
defun, 1063
loopIters2Sex, 309
calledby pf2Sex1, 298
calledby pfCollect2Sex, 312
calls pf2Sex1, 310
defun, 309
lotsof, 1037
calledby wrap, 1037
defun, 1037
ltrace, 628
calls trace, 628
defun, 628
ltrace help page, 627
manpage, 627
lxOK1
calledby xlOK, 86

mac0Define, 228

calledby mac0MLambdaApply, 222
calledby macMacro, 227
uses $pfMacros, 228
defun, 228
mac0ExpandBody, 222
calledby mac0MLambdaApply, 222
calledby macId, 225
calls mac0InfiniteExpansion, 222
calls macExpand, 223
calls pfSourcePosition, 222
uses $macActive, 223
uses $posActive, 223
defun, 222
mac0Get, 226
calledby macId, 225
calls ifcdr, 226
uses $pfMacros, 226
defun, 226
mac0GetName, 224
calledby mac0InfiniteExpansion,name, 224
calls pfMLambdaBody, 224
uses $pfMacros, 224
defun, 224
mac0InfiniteExpansion, 223
calledby mac0ExpandBody, 222
calls mac0InfiniteExpansion,name, 223
calls ncSoftError, 223
calls pform, 223
defun, 223
mac0InfiniteExpansion,name, 224
calledby mac0InfiniteExpansion, 223
calls mac0GetName, 224
calls pname, 224
defun, 224
mac0MLambdaApply, 221
calledby macApplication, 221
calls mac0Define, 222
calls mac0ExpandBody, 222
calls ncHardError, 221
calls pf0MLambdaArgs, 221
calls pfId?, 222
calls pfMLambdaBody, 221
calls pfSourcePosition, 221
calls pform, 222
uses $macActive, 222
uses $pfMacros, 222
uses $posActive, 222
defun, 221
mac0SubstituteOuter, 229
calledby mac0SubstituteOuter, 229
calledby macSubstituteOuter, 228
calls mac0SubstituteOuter, 229
calls macLambdaParameterHandling, 229
calls macSubstituteId, 229
calls pfId?, 229
calls pfLambda?, 229
calls pfLeaf?, 229
calls pfParts, 229
defun, 229
macApplication, 221
calledby macExpand, 220
calls mac0MLambdaApply, 221
calls macExpand, 221
calls pf0ApplicationArgs, 221
calls pfApplicationOp, 221
calls pfMLambda?, 221
calls pfMapParts, 221
uses $pfMacros, 221
defun, 221
macExpand, 220
calledby mac0ExpandBody, 223
calledby macApplication, 221
calledby macExpand, 220
calledby macLambda,mac, 227
calledby macWhere,mac, 226
calledby macroExpanded, 220
calls macApplication, 220
calls macExpand, 220
calls macId, 220
calls macLambda, 220
calls macMacro, 220
calls macWhere, 220
calls pfApplication?, 220
calls pfId?, 220
calls pfLambda?, 220
calls pfMacro?, 220
calls pfMapParts, 220
calls pfWhere?, 220
defun, 220
macId, 225
calledby macExpand, 220
calls mac0ExpandBody, 225

calls mac0Get, 225
calls pfCopyWithPos, 225
calls pfIdSymbol, 225
calls pfSourcePosition, 225
uses $macActive, 225
uses $posActive, 225
defun, 225
macLambda, 226
calledby macExpand, 220
calls macLambda,mac, 226
uses $pfMacros, 227
defun, 226
macLambda,mac, 227
calledby macLambda, 226
calls macExpand, 227
calls pfMapParts, 227
uses $pfMacros, 227
defun, 227
macLambdaParameterHandling, 229
calledby mac0SubstituteOuter, 229
calledby macLambdaParameterHandling, 229
calledby macSubstituteOuter, 228
calls macLambdaParameterHandling, 229
calls pf0LambdaArgs, 229
calls pf0MLambdaArgs, 229
calls pfAbSynOp, 229
calls pfIdSymbol, 229
calls pfLambda?, 229
calls pfLeaf?, 229
calls pfLeafPosition, 229
calls pfLeaf, 229
calls pfMLambda?, 229
calls pfParts, 229
calls pfTypedId, 229
defun, 229
macMacro, 227
calledby macExpand, 220
calls mac0Define, 227
calls macSubstituteOuter, 227
calls ncSoftError, 227
calls pfId?, 227
calls pfIdSymbol, 227
calls pfMLambda?, 227
calls pfMacroLhs, 227
calls pfMacroRhs, 227
calls pfMacro, 227
calls pfNothing?, 227
calls pfNothing, 227
calls pfSourcePosition, 227
calls pform, 227
defun, 227
macroExpanded, 220
calledby parseFromString, 48
calledby phMacro, 219
calls macExpand, 220
uses $macActive, 220
uses $posActive, 220
defun, 220
macSubstituteId, 230
calledby mac0SubstituteOuter, 229
calls pfIdSymbol, 230
defun, 230
macSubstituteOuter, 228
calledby macMacro, 227
calls mac0SubstituteOuter, 228
calls macLambdaParameterHandling, 228
defun, 228
macWhere, 226
calledby macExpand, 220
calls macWhere,mac, 226
uses $pfMacros, 226
defun, 226
macWhere,mac, 226
calledby macWhere, 226
calls macExpand, 226
calls pfMapParts, 226
uses $pfMacros, 226
defun, 226
make-absolute-filename, 36
calledby reroot, 41
uses $spadroot, 36
defun, 36
make-appendstream, 972
calledby makeStream, 973
calls make-filename, 972
defun, 972
make-cdouble-matrix, 1070
defmacro, 1070
make-cdouble-vector, 1067
defmacro, 1067
make-database

calledby interpOpen, 995
calledby localnrlib, 1007
calledby setdatabase, 1000
make-databases, 1009
calls allConstructors, 1010
calls browserAutoloadOnceTrigger, 1010
calls buildLibdb, 1010
calls categoryForm?, 1010
calls dbSplitLibdb, 1010
calls domainsOf, 1010
calls getConstructorForm, 1010
calls getEnv, 1010
calls localdatabase, 1010
calls mkDependentsHashTable, 1010
calls mkTopicHashTable, 1010
calls mkUsersHashTable, 1010
calls saveDependentsHashTable, 1010
calls saveUsersHashTable, 1010
calls write-browsedb, 1010
calls write-categorydb, 1010
calls write-compress, 1010
calls write-interpdb, 1010
calls write-operationdb, 1010
calls write-warmdata, 1010
uses *allconstructors*, 1010
uses *compressvector*, 1010
uses *operation-hash*, 1010
uses *sourcefiles*, 1010
uses $constructorList, 1010
defun, 1009
make-double-matrix, 1069
defmacro, 1069
make-double-matrix1, 1069
defmacro, 1069
make-double-vector, 1066
defmacro, 1066
make-double-vector1, 1066
defmacro, 1066
make-filename
calledby make-appendstream, 972
calledby make-outstream, 971
calledby pathname, 1036
make-instream, 971
calledby dewritify,dewritifyInner, 608
calledby incConsoleInput, 102
calledby incFileInput, 102
calledby newHelpSpad2Cmd, 569
calls makeInputFilename, 971
defun, 971
make-monitor-data
calledby monitor-add, 1150
make-outstream, 971
calledby makeStream, 973
calledby setOutputAlgebra, 754
calledby setOutputFormula, 782
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calls make-filename, 971
defun, 971
make-string
calledby executeQuietCommand, 47
makeByteWordVec2, 1139
defun, 1139
makeFullNamestring, 975
calledby makeStream, 975
defun, 975
makeHistFileName, 575
calledby closeInterpreterFrame, 558
calledby histFileName, 576
calledby oldHistFileName, 576
calledby restoreHistory, 593
calledby saveHistory, 591
calls makePathname, 575
defun, 575
makeInitialModemapFrame, 37
calledby restart, 17
calls copy, 37
uses $InitialModemapFrame, 37
defun, 37
makeInputFilename, 973
calledby initHist, 577
calledby make-instream, 971
calledby newHelpSpad2Cmd, 568
calledby restoreHistory, 593
calledby saveHistory, 591
calledby updateSourceFiles, 542
calledby workfilesSpad2Cmd, 939
defun, 973
makeLeaderMsg, 375
calledby processChPosesForOneLine, 374

uses $nopos, 375
uses $preLength, 375
defun, 375
makeLongSpaceString
calledby printStorage, 58
makeLongTimeString
calledby printTypeAndTimeNormal, 59
calledby printTypeAndTimeSaturn, 60
makeMsgFromLine, 369
calledby processMsgList, 367
calls char, 369
calls getLinePos, 369
calls getLineText, 369
calls poGlobalLinePosn, 369
calls poLinePosn, 369
calls rep, 369
calls size, 369
calls strconc, 369
uses $preLength, 369
defun, 369
makeOrdinal, 910
calledby evaluateType1, 908
defun, 910
makePathname, 1036
calledby histInputFileName, 576
calledby makeHistFileName, 575
calledby readSpad2Cmd, 638
calledby sayMSG2File, 329
calls object2String, 1036
calls pathname, 1036
defun, 1036
makeStream, 973
calledby setOutputFortran, 761
calls make-appendstream, 973
calls make-outstream, 973
calls makeFullNamestring, 975
defun, 973
MakeSymbol
calledby assertCond, 96
calledby ifCond, 89
manexp, 1088
defun, 1088
mapage
abbreviations help page, 477
boot help page, 483
browse help page, 485
cd help page, 491
clear help page, 493
close help page, 505
compile help page, 509
copyright help page, 513
credits help page, 521
describe help page, 523
display help page, 529
edit help page, 539
fin help page, 543
frame help page, 545
help help page, 565
history help page, 571
include help page, 617
library help page, 621
lisp help page, 623
load help page, 625
ltrace help page, 627
pquit help page, 629
quit help page, 633
read help page, 637
savesystem help page, 641
set help page, 643
show help page, 805
spool help page, 819
summary help page, 821
synonym help page, 823
system help page, 829
trace help page, 831
undo help page, 899
what help page, 927
with help page, 937
workfiles help page, 939
zsystemdevelopment help page, 943
mapLetPrint, 874
calls getAliasIfTracedMapParameter, 874
calls getBpiNameIfTracedMap, 874
calls letPrint, 874
defun, 874
mathprint
calledby displayMacro, 450
calledby displayValue, 455
calledby letPrint2, 877
calledby letPrint3, 878
maxindex

calledby processSynonymLine,removeKeyFromLine, 826
calledby removeUndoLines, 924
member, 1042
calledby clearCmdParts, 501
calledby displayMacros, 534
calledby displayProperties, 458
calledby doSystemCommand, 442
calledby evaluateType, 906
calledby fixObjectForPrinting, 452
calledby getBrowseDatabase, 1075
calledby handleNoParseCommands, 466
calledby hasCateSpecialNew, 433
calledby historySpad2Cmd, 578
calledby importFromFrame, 559
calledby interpret2, 55
calledby isDomainValuedVariable, 949
calledby isSubForRedundantMapName, 863
calledby readSpad2Cmd, 638
calledby reportOpsFromUnitDirectly, 813
calledby selectOption, 475
calledby setExposeAddConstr, 692
calledby setExposeAddGroup, 691
calledby setExposeDropConstr, 695
calledby setExposeDropGroup, 694
calledby setOutputAlgebra, 754
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby showSpad2Cmd, 806
calledby transTraceItem, 853
calledby translateYesNo2TrueFalse, 650
calledby updateSourceFiles, 542
defun, 1042
mergePathnames, 1035
calledby readSpad2Cmd, 638
calls pathnameDirectory, 1035
calls pathnameName, 1035
calls pathnameType, 1035
defun, 1035
messageprint, 1042
calledby brightprint, 1041
defun, 1042
messageprint-1, 1043
calledby brightprint-0, 1042
calledby messageprint-1, 1043
calledby messageprint-2, 1043
calls identp, 1043
calls messageprint-1, 1043
calls messageprint-2, 1043
defun, 1043
messageprint-2, 1043
calledby messageprint-1, 1043
calledby messageprint-2, 1043
calls messageprint-1, 1043
calls messageprint-2, 1043
defun, 1043
meta-error-handler
usedby init-boot/spad-reader, 958
mkAtree
calledby evaluateType1, 908
calledby evaluateType, 906
calledby reportOperations, 808
mkAtreeWithSrcPos
calledby interpret1, 54
mkCompanionPage
calledby recordAndPrint, 56
mkDependentsHashTable
calledby make-databases, 1010
mkDomPvar, 430
calledby hasCaty, 418
calls domArg, 430
calls length, 430
local ref $FormalMapVariableList, 431
defun, 430
mkEvalable, 903
calledby evalDomain, 903
calledby mkEvalableMapping, 905
calledby mkEvalableRecord, 905
calledby mkEvalableUnion, 905
calledby mkEvalable, 903
calledby writify,writifyInner, 603
calls bpiname, 903
calls constructor?, 903
calls devaluate, 903
calls fbpip, 903
calls getdatabase, 903
calls loadIfNecessary, 903
calls mkEvalableMapping, 903

calls mkEvalableRecord, 903
calls mkEvalableUnion, 903
calls mkEvalable, 903
calls mkq, 903
calls qcar, 903
calls qcdr, 903
local ref $EmptyMode, 903
local ref $Integer, 903
defun, 903
mkEvalableMapping, 905
calledby mkEvalable, 903
calls mkEvalable, 905
defun, 905
mkEvalableRecord, 905
calledby mkEvalable, 903
calls mkEvalable, 905
defun, 905
mkEvalableUnion, 905
calledby mkEvalable, 903
calls mkEvalable, 905
defun, 905
mkLineList, 70
calledby intloopEchoParse, 69
defun, 70
mkprompt, 42
calledby SpadInterpretStream, 29
calledby intloopReadConsole, 30
calledby serverReadLine, 44
calls concat, 42
calls currenttime, 42
calls substring, 42
uses $IOindex, 42
uses $inputPromptType, 42
uses $interpreterFrameName, 42
defun, 42
mkq
calledby mkEvalable, 903
calledby traceDomainConstructor, 871
mkTopicHashTable
calledby make-databases, 1010
mkUserConstructorAbbreviation
calledby abbreviationsSpad2Cmd, 479
mkUsersHashTable
calledby make-databases, 1010
modes
calledby clearCmdParts, 501
MONITOR,EVALTRAN
calledby break, 896
monitor-add, 1150
calledby monitor-file, 1153
calls make-monitor-data, 1150
calls monitor-delete, 1150
uses *monitor-table*, 1150
defun, 1150
monitor-apropos, 1163
uses *monitor-table*, 1163
defun, 1163
monitor-autoload, 1158
defun, 1158
monitor-checkpoint, 1155
uses *monitor-table*, 1155
uses *print-package*, 1155
defun, 1155
monitor-data, 1148
defstruct, 1148
monitor-decr, 1152
uses *monitor-table*, 1152
defun, 1152
monitor-delete, 1150
calledby monitor-add, 1150
calledby monitor-tested, 1154
uses *monitor-table*, 1150
defun, 1150
monitor-dirname, 1158
uses *monitor-nrlibs*, 1158
defun, 1158
monitor-disable, 1151
uses *monitor-table*, 1151
defun, 1151
monitor-enable, 1150
uses *monitor-table*, 1150
defun, 1150
monitor-end, 1149
uses *monitor-table*, 1149
defun, 1149
monitor-exposedp, 1160
defun, 1160
monitor-file, 1153
calls monitor-add, 1153
catches, 1153
defun, 1153
throws, 1153

monitor-help, 1156
defun, 1156
monitor-incr, 1152
uses *monitor-table*, 1152
defun, 1152
monitor-info, 1153
uses *monitor-table*, 1153
defun, 1153
monitor-inittable, 1148
uses *monitor-table*, 1148
defun, 1148
monitor-libname, 1159
defun, 1159
monitor-nrlib, 1159
uses *monitor-table*, 1159
defun, 1159
monitor-parse, 1162
calledby monitor-spadfile, 1162
defun, 1162
monitor-percent, 1163
uses *monitor-table*, 1163
defun, 1163
monitor-readinterp, 1160
calledby monitor-report, 1161
uses *monitor-domains*, 1160
catches, 1160
defun, 1160
throws, 1160
monitor-report, 1161
calls monitor-readinterp, 1161
uses *monitor-domains*, 1161
defun, 1161
monitor-reset, 1151
uses *monitor-table*, 1151
defun, 1151
monitor-restore, 1155
defun, 1155
monitor-results, 1149
uses *monitor-table*, 1149
defun, 1149
monitor-spadfile, 1162
calls monitor-parse, 1162
uses *monitor-domains*, 1162
catches, 1162
defun, 1162
throws, 1162
monitor-tested, 1154
calls monitor-delete, 1154
uses *monitor-table*), 1154
defun, 1154
monitor-untested, 1153
uses *monitor-table*, 1153
defun, 1153
monitor-write, 1154
defun, 1154
msgCreate, 345
calledby ncBug, 366
calledby ncHardError, 350
calledby ncSoftError, 349
calls initImPr, 346
calls initToWhere, 346
calls putDatabaseStuff, 346
calls setMsgForcedAttrList, 346
defun, 345
msgImPr?, 356
calledby alreadyOpened?, 361
calledby getPosStL, 354
calledby processKeyedError, 351
calledby showMsgPos?, 355
calls getMsgCatAttr, 356
defun, 356
msgNoRep?, 373
calledby redundant, 372
calls getMsgCatAttr, 373
defun, 373
msgOutputter, 351
calledby listOutputter, 352
calledby processKeyedError, 351
calls alreadyOpened?, 351
calls flowSegmentedMsg, 351
calls getStFromMsg, 351
calls leader?, 351
calls line?, 351
calls sayBrightly, 351
calls toFile?, 351
calls toScreen?, 351
uses $linelength, 351
defun, 351
msgText, 61
calledby printTypeAndTimeNormal, 59
calls flowSegmentedMsg, 62
calls getKeyedMsg, 62

calls segmentKeyedMsg, 61
calls substituteSegmentedMsg, 62
uses $linelength, 62
uses $margin, 62
defun, 61
msort
calledby apropos, 935
calledby displayOperationsFromLisplib, 812
calledby displayProperties, 457
calledby displayWorkspaceNames, 450
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calledby saveDependentsHashTable, 1013
calledby saveUsersHashTable, 1014
calledby setExposeAddConstr, 692
calledby setExposeAddGroup, 691
calledby setExposeDropConstr, 695
calledby whatConstructors, 935
myWritable?, 1078
calls error, 1078
calls fnameDirectory, 1078
calls fnameExists?, 1078
calls writeablep, 1078
defun, 1078
myWriteable?
calledby fnameWritable?, 1078

names
calledby frameSpad2Cmd, 562
namestring, 1034
calledby editFile, 541
calledby loadLib, 1029
calledby newHelpSpad2Cmd, 569
calledby readSpad2Cmd, 638
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calledby restoreHistory, 593
calledby saveHistory, 592
calledby setExposeAddGroup, 691
calledby setExpose, 689
calledby workfilesSpad2Cmd, 939
calledby writeInputLines, 583
calls pathname, 1034
defun, 1034
ncAbort
calledby ncBug, 366
ncAlist, 413
calledby getMsgCatAttr, 356
calledby ncEltQ, 414
calledby ncPutQ, 415
calledby setMsgCatlessAttr, 363
calledby setMsgUnforcedAttr, 365
calledby tokPosn, 411
calls identp, 413
calls ncBug, 413
calls qcar, 413
calls qcdr, 414
defun, 413
ncBug, 366
calledby getMsgPos2, 377
calledby incHandleMessage, 76
calledby lnFileName, 345
calledby ncAlist, 413
calledby ncEltQ, 414
calledby ncTag, 413
calledby poGlobalLinePosn, 72
calls enable-backtrace, 366
calls msgCreate, 366
calls ncAbort, 366
calls processKeyedError, 366
uses $newcompErrorCount, 366
uses $nopos, 366
defun, 366
ncConversationPhase, 68
calledby intloopSpadProcess,interp, 65
calls ncConversationPhase,wrapup, 68
uses $ncMsgList, 68
defun, 68
ncConversationPhase,wrapup, 68
calledby ncConversationPhase, 68
uses $ncMsgList, 68
defun, 68
ncEltQ, 414
calledby intloopSpadProcess,interp, 66
calledby phIntReportMsgs, 66
calledby phInterpret, 67
calledby phMacro, 219
calls ncAlist, 414
calls ncBug, 414
calls qassq, 414
defun, 414

ncError, 69
  calledby intloopSpadProcess,interp, 66
  calledby ncHardError, 350
  defun, 69
  throws, 69
ncHardError, 350
  calledby mac0MLambdaApply, 221
  calls desiredMsg, 350
  calls msgCreate, 350
  calls ncError, 350
  calls processKeyedError, 350
  uses $newcompErrorCount, 350
  defun, 350
ncIntLoop, 25
  calledby ncTopLevel, 25
  calls intloop, 25
  uses curinstream, 25
  uses curoutstream, 26
  defun, 25
ncloopCommand, 474
  calledby intloopReadConsole, 30
  calls $systemCommandFunction, 474
  calls ncloopInclude1, 474
  calls ncloopPrefix?, 474
  uses $systemCommandFunction, 474
  defun, 474
ncloopDQlines, 71
  calledby intloopEchoParse, 69
  calledby ncloopParse, 38
  calls StreamNull, 71
  calls poGlobalLinePosn, 71
  calls streamChop, 71
  calls tokPosn, 71
  defun, 71
ncloopEchoParse
  calledby ncloopInclude0, 73
ncloopEscaped, 37
  calledby intloopReadConsole, 30
  defun, 37
ncloopIncFileName, 618
  calledby ncloopInclude1, 617
  calls concat, 618
  calls incFileName, 618
  defun, 618
ncloopInclude, 618
  calledby ncloopInclude1, 617
  calls ncloopInclude0, 618
  defun, 618
ncloopInclude0, 73
  calledby ncloopInclude, 618
  calls incStream, 73
  calls insertpile, 73
  calls lineoftoks, 73
  calls ncloopEchoParse, 73
  calls ncloopProcess, 73
  calls next, 73
  uses $lines, 73
  defun, 73
ncloopInclude1, 617
  calledby ncloopCommand, 474
  calls ncloopIncFileName, 617
  calls ncloopInclude, 617
  defun, 617
ncloopParse, 38
  calledby parseFromString, 48
  calls dqToList, 38
  calls ncloopDQlines, 38
  calls npParse, 38
  defun, 38
ncloopPrefix?, 475
  calledby ncloopCommand, 474
  calledby streamChop, 72
  defun, 475
ncloopPrintLines, 70
  calledby intloopEchoParse, 69
  defun, 70
ncloopProcess
  calledby ncloopInclude0, 73
ncParseFromString, 1052
  defun, 1052
ncPutQ, 414
  calledby constoken, 111
  calledby intloopSpadProcess, 65
  calledby phIntReportMsgs, 66
  calledby phInterpret, 67
  calledby phMacro, 219
  calledby phParse, 66
  calledby setMsgCatlessAttr, 363
  calledby setMsgForcedAttr, 362
  calledby setMsgUnforcedAttr, 365
  calledby tokConstruct, 409
  calls ncAlist, 415

calls ncTag, 415
calls qassq, 415
defun, 414
ncSoftError, 349
calledby incHandleMessage, 76
calledby mac0InfiniteExpansion, 223
calledby macMacro, 227
calledby npMissingMate, 213
calledby npMissing, 149
calledby npParse, 139
calledby npTrapForm, 210
calledby npTrap, 210
calledby scanError, 132
calledby scanS, 128
calledby syIgnoredFromTo, 189
calledby sySpecificErrorAtToken, 190
calls desiredMsg, 349
calls msgCreate, 349
calls processKeyedError, 349
uses $newcompErrorCount, 349
defun, 349
ncTag, 413
calledby getMsgTag, 348
calledby ncPutQ, 415
calledby tokType, 411
calls identp, 413
calls ncBug, 413
calls qcar, 413
defun, 413
ncTopLevel, 25
calledby runspad, 21
calls ncIntLoop, 25
uses *eof*, 25
uses $InteractiveFrame, 25
uses $InteractiveMode, 25
uses $boot, 25
uses $e, 25
uses $newspad, 25
uses $spad, 25
uses in-stream, 25
defun, 25
new
calledby frameSpad2Cmd, 562
newHelpSpad2Cmd, 568
calledby helpSpad2Cmd, 568
calls concat, 568
calls make-instream, 569
calls makeInputFilename, 568
calls namestring, 569
calls obey, 568
calls pname, 569
calls poundsign, 569
calls sayKeyedMsg, 569
calls say, 569
calls selectOptionLC, 569
uses $syscommands, 569
uses $useFullScreenHelp, 569
defun, 568
next, 38
calledby frameSpad2Cmd, 562
calledby intloopInclude0, 63
calledby intloopProcessString, 37
calledby ncloopInclude0, 73
calledby next1, 39
calledby parseFromString, 48
calledby zsystemdevelopment1, 944
calls Delay, 38
calls next1, 38
defun, 38
next-lines-clear, 962
calledby init-boot/spad-reader, 958
uses boot-line-stack, 962
defun, 962
next-lines-show, 961
calledby iostat, 960
uses boot-line-stack, 961
defun, 961
next-token
usedby token-stack-show, 961
next1, 38
calledby next, 38
calls StreamNull, 38
calls incAppend, 39
calls next, 39
defun, 38
nextInterpreterFrame, 556
calledby frameSpad2Cmd, 562
calls updateFromCurrentInterpreterFrame, 556
uses $interpreterFrameRing, 556
defun, 556
nextline, 110

calledby lineoftoks, 109
calledby scanEsc, 121
calls npNull, 110
calls strposl, 110
uses $f, 110
uses $linepos, 110
uses $ln, 110
uses $n, 110
uses $r, 110
uses $sz, 110
defun, 110
nmsort
calledby getWorkspaceNames, 451
nonBlank, 71
defun, 71
npADD, 156
calledby npExpress1, 177
calls npAdd, 156
calls npPop1, 156
calls npPush, 156
calls npType, 156
defun, 156
npAdd, 157
calledby npADD, 156
calledby npPrimary2, 156
calls npCompMissing, 157
calls npDefinitionOrStatement, 157
calls npEqKey, 157
calls npEqPeek, 157
calls npPop1, 157
calls npPop2, 157
calls npPush, 157
calls npRestore, 157
calls npState, 157
calls npTrap, 157
calls npVariable, 157
calls pfAdd, 157
calls pfNothing, 157
defun, 157
npAmpersand, 202
calledby npAmpersandFrom, 200
calledby npAtom2, 158
calls npEqKey, 202
calls npName, 202
calls npTrap, 202
defun, 202
npAmpersandFrom, 200
calledby npSynthetic, 196
calls npAmpersand, 200
calls npFromdom, 200
defun, 200
npAndOr, 179
calledby npImport, 178
calledby npInline, 172
calledby npSuchThat, 174
calledby npVoid, 177
calledby npWhile, 175
calls npEqKey, 179
calls npPop1, 180
calls npPush, 180
calls npTrap, 180
defun, 179
npAngleBared, 184
calledby npBracketed, 183
calls npEnclosed, 184
calls pfHide, 184
defun, 184
npAnyNo, 160
calledby npColon, 215
calledby npEncAp, 180
calledby npTypified, 216
defun, 160
npApplication, 160
calledby npFromdom1, 201
calledby npFromdom, 201
calledby npSCategory, 151
calledby npTypedForm, 218
calledby npTypified, 216
calls npApplication2, 160
calls npDotted, 160
calls npPop1, 160
calls npPop2, 160
calls npPrimary, 160
calls npPush, 160
calls pfApplication, 160
defun, 160
npApplication2, 161
calledby npApplication2, 161
calledby npApplication, 160
calls npApplication2, 161
calls npDotted, 161
calls npPop1, 161

calls npPop2, 161
calls npPrimary1, 161
calls npPush, 161
calls pfApplication, 161
defun, 161
npArith, 198
calledby npInterval, 197
calls npLeftAssoc, 198
calls npSum, 198
defun, 198
npAssign, 214
calledby npExit, 214
calledby npLoop, 173
calledby npPileExit, 214
calls npAssignment, 214
calls npBackTrack, 214
calls npMDEF, 214
defun, 214
npAssignment, 215
calledby npAssign, 214
calls npAssignVariable, 215
calls npEqKey, 215
calls npGives, 215
calls npPop1, 215
calls npPop2, 215
calls npPush, 215
calls npTrap, 215
calls pfAssign, 215
defun, 215
npAssignVariable, 215
calledby npAssignment, 215
calls npColon, 215
calls npPop1, 215
calls npPush, 215
calls pfListOf, 215
defun, 215
npAtom1, 181
calledby npPrimary1, 162
calls npBDefinition, 181
calls npConstTok, 181
calls npDollar, 181
calls npFromdom, 181
calls npName, 181
calls npPDefinition, 181
defun, 181
npAtom2, 157
calledby npPrimary2, 156
calls npAmpersand, 158
calls npFromdom, 158
calls npInfixOperator, 157
calls npPrefixColon, 158
defun, 157
npBacksetElse, 195
calledby npElse, 194
calls npEqKey, 195
defun, 195
npBackTrack, 146
calledby npAssign, 214
calledby npDefinitionOrStatement, 145
calledby npExit, 213
calledby npGives, 146
calledby npMDEF, 163
calls npEqPeek, 146
calls npRestore, 146
calls npState, 146
calls npTrap, 146
defun, 146
npBDefinition, 183
calledby npAtom1, 181
calledby npEncl, 180
calls npBracketed, 183
calls npDefinitionlist, 183
calls npPDefinition, 183
defun, 183
npboot, 468
calledby handleNoParseCommands, 466
defun, 468
npBPileDefinition, 186
calledby npPrimary1, 162
calls npPileBracketed, 186
calls npPileDefinitionlist, 186
calls npPop1, 186
calls npPush, 186
calls pfListOf, 186
calls pfSequence, 186
defun, 186
npBraced, 184
calledby npBracketed, 183
calls npEnclosed, 184
calls pfBraceBar, 184
calls pfBrace, 184
defun, 184

npBracked, 184
calledby npBracketed, 183
calls npEnclosed, 184
calls pfBracketBar, 184
calls pfBracket, 184
defun, 184
npBracketed, 183
calledby npBDefinition, 183
calls npAngleBared, 183
calls npBraced, 183
calls npBracked, 183
calls npParened, 183
defun, 183
npBreak, 172
calledby npStatement, 168
calls npEqKey, 172
calls npPush, 172
calls pfBreak, 172
calls pfNothing, 172
defun, 172
npBy, 197
calledby npForIn, 175
calledby npSynthetic, 196
calls npInterval, 197
calls npLeftAssoc, 197
defun, 197
npCategory, 151
calledby npCategoryL, 150
calls npPP, 151
calls npSCategory, 151
defun, 151
npCategoryL, 150
calledby npSCategory, 151
calledby npWith, 148
calls npCategory, 150
calls npPop1, 150
calls npPush, 150
calls pfUnSequence, 150
defun, 150
npCoerceTo, 218
calledby npTypeStyle, 217
calls npTypedForm, 218
calls pfCoerceto, 218
defun, 218
npColon, 215
calledby npAssignVariable, 215
calledby npPower, 200
calls npAnyNo, 215
calls npTagged, 215
calls npTypified, 215
defun, 215
npColonQuery, 217
calledby npTypeStyle, 217
calls npTypedForm, 217
calls pfRetractTo, 217
defun, 217
npComma, 144
calledby npQualDef, 143
calls npQualifiedDefinition, 144
calls npTuple, 144
defun, 144
npCommaBackSet, 144
calledby npTuple, 144
calls npEqKey, 144
defun, 144
npCompMissing, 149
calledby npAdd, 157
calledby npForIn, 175
calledby npLetQualified, 164
calledby npLoop, 173
calledby npWith, 148
calls npEqKey, 149
calls npMissing, 149
defun, 149
npConditional, 193
calledby npConditionalStatement, 178
calledby npWConditional, 193
calls npElse, 193
calls npEqKey, 193
calls npLogical, 193
calls npMissing, 193
calls npTrap, 193
defun, 193
npConditionalStatement, 178
calledby npExpress1, 177
calls npConditional, 178
calls npQualifiedDefinition, 178
defun, 178
npConstTok, 182
calledby npAtom1, 181
calls npEqPeek, 182
calls npNext, 182

calls npPop1, 182
calls npPrimary1, 182
calls npPush, 182
calls npRestore, 182
calls npState, 182
calls pfSymb, 182
calls tokPosn, 182
calls tokType, 182
uses $stok, 182
defun, 182
npDDInfKey, 206
calledby npInfGeneric, 205
calls npEqKey, 206
calls npInfKey, 206
calls npPop1, 206
calls npPush, 206
calls npRestore, 206
calls npState, 206
calls pfSymb, 206
calls tokConstruct, 206
calls tokPart, 206
calls tokPosn, 206
uses $stok, 206
defun, 206
npDecl, 212
calledby npVariableName, 212
calls npEqKey, 212
calls npPop1, 212
calls npPop2, 212
calls npPush, 212
calls npTrap, 212
calls npType, 212
calls pfTyped, 212
defun, 212
npDef, 185
calledby npDefinitionItem, 165
calledby npDefinitionOrStatement, 145
calledby npDefn, 185
calledby npFix, 164
calls npDefTail, 185
calls npMatch, 185
calls npPop1, 185
calls npPush, 185
calls npTrap, 185
calls pfCheckItOut, 185
calls pfDefinition, 185
calls pfPushBody, 185
defun, 185
npDefaultDecl, 168
calledby npDefaultItem, 167
calls npEqKey, 168
calls npPop1, 168
calls npPop2, 168
calls npPush, 168
calls npTrap, 168
calls npType, 168
calls pfParts, 168
calls pfSpread, 168
defun, 168
npDefaultItem, 167
calledby npSDefaultItem, 167
calls npDefaultDecl, 167
calls npTrap, 167
calls npTypeVariable, 167
defun, 167
npDefaultItemlist, 166
calledby npTyping, 166
calls npPC, 166
calls npPop1, 167
calls npPush, 167
calls npSDefaultItem, 167
calls pfUnSequence, 167
defun, 166
npDefaultValue, 192
calledby npSCategory, 151
calls npDefinitionOrStatement, 193
calls npEqKey, 192
calls npPop1, 193
calls npPush, 193
calls npTrap, 193
calls pfAdd, 193
calls pfNothing, 193
defun, 192
npDefinition, 165
calledby npLetQualified, 164
calledby npQualified, 145
calls npDefinitionItem, 165
calls npPP, 165
calls npPop1, 165
calls npPush, 165
calls pfSequenceToList, 165
defun, 165

npDefinitionItem, 165
calledby npDefinition, 165
calls npDefn, 166
calls npDef, 165
calls npEqPeek, 165
calls npImport, 165
calls npMacro, 165
calls npRestore, 165
calls npStatement, 165
calls npState, 165
calls npTrap, 166
calls npTyping, 165
defun, 165
npDefinitionlist, 191
calledby npBDefinition, 183
calledby npPDefinition, 181
calledby npPileDefinitionlist, 187
calls npQualDef, 191
calls npSemiListing, 191
defun, 191
npDefinitionOrStatement, 145
calledby npAdd, 157
calledby npDefTail, 192
calledby npDefaultValue, 193
calledby npLambda, 147
calledby npLet, 164
calledby npQualifiedDefinition, 145
calls npBackTrack, 145
calls npDef, 145
calls npGives, 145
defun, 145
npDefn, 185
calledby npDefinitionItem, 166
calledby npPrimary1, 162
calls npDef, 185
calls npEqKey, 185
calls npPP, 185
defun, 185
npDefTail, 192
calledby npDef, 185
calledby npMdef, 163
calledby npSingleRule, 192
calls npDefinitionOrStatement, 192
calls npEqKey, 192
defun, 192
npDiscrim, 195
calledby npDisjand, 195
calls npLeftAssoc, 195
calls npQuiver, 195
defun, 195
npDisjand, 195
calledby npLogical, 195
calls npDiscrim, 195
calls npLeftAssoc, 195
defun, 195
npDollar, 181
calledby npAtom1, 181
calls npEqPeek, 181
calls npNext, 182
calls npPush, 181
calls tokConstruct, 181
calls tokPosn, 181
uses $stok, 182
defun, 181
npDotted, 160
calledby npApplication2, 161
calledby npApplication, 160
calls , 160
defun, 160
npElse, 194
calledby npConditional, 193
calls npBacksetElse, 194
calls npPop1, 194
calls npPop2, 194
calls npPop3, 194
calls npPush, 194
calls npRestore, 194
calls npState, 194
calls npTrap, 194
calls pfIfThenOnly, 194
calls pfIf, 194
defun, 194
npEncAp, 180
calledby npPrimary1, 162
calledby npPrimary2, 156
calls npAnyNo, 180
calls npEncl, 180
calls npFromdom, 180
defun, 180
npEncl, 180
calledby npEncAp, 180
calls npBDefinition, 180

calls npPop1, 180
calls npPop2, 180
calls npPush, 180
calls pfApplication, 180
defun, 180
npEnclosed, 209
calledby npAngleBared, 184
calledby npBraced, 184
calledby npBracked, 184
calledby npParened, 183
calls npEqKey, 209
calls npMissingMate, 209
calls npPop1, 209
calls npPush, 209
calls pfEnSequence, 209
calls pfListOf, 209
calls pfTuple, 209
uses $stok, 209
defun, 209
npEqKey, 143
calledby npAdd, 157
calledby npAmpersand, 202
calledby npAndOr, 179
calledby npAssignment, 215
calledby npBacksetElse, 195
calledby npBreak, 172
calledby npCommaBackSet, 144
calledby npCompMissing, 149
calledby npConditional, 193
calledby npDDInfKey, 206
calledby npDecl, 212
calledby npDefTail, 192
calledby npDefaultDecl, 168
calledby npDefaultValue, 192
calledby npDefn, 185
calledby npEnclosed, 209
calledby npExport, 169
calledby npFix, 164
calledby npForIn, 175
calledby npFree, 171
calledby npFromdom1, 201
calledby npFromdom, 200
calledby npInfGeneric, 205
calledby npInfixOperator, 158
calledby npItem1, 140
calledby npItem, 140
calledby npIterate, 172
calledby npLambda, 147
calledby npLetQualified, 164
calledby npListAndRecover, 187
calledby npList, 153
calledby npLocalDecl, 170
calledby npLocal, 171
calledby npLoop, 173
calledby npMoveTo, 189
calledby npParenthesize, 213
calledby npPileBracketed, 186
calledby npPileExit, 214
calledby npQualified, 145
calledby npReturn, 176
calledby npRule, 191
calledby npSelector, 161
calledby npSemiBackSet, 191
calledby npSigDecl, 155
calledby npSymbolVariable, 203
calledby npTypedForm1, 216
calledby npTypedForm, 218
calledby npTyping, 166
calledby npWith, 148
calls npNext, 143
uses $stok, 143
uses $ttok, 143
defun, 143
npEqPeek, 150
calledby npAdd, 157
calledby npBackTrack, 146
calledby npConstTok, 182
calledby npDefinitionItem, 165
calledby npDollar, 181
calledby npInterval, 197
calledby npListAndRecover, 187
calledby npMoveTo, 189
calledby npPrefixColon, 159
calledby npSCategory, 151
calledby npSegment, 198
calledby npWith, 148
uses $stok, 150
uses $ttok, 150
defun, 150
npExit, 213
calledby npGives, 146
calls npAssign, 214

calls npBackTrack, 213
calls npPileExit, 214
defun, 213
npExport, 169
calledby npStatement, 168
calls npEqKey, 169
calls npLocalItemlist, 169
calls npPop1, 169
calls npPush, 169
calls npTrap, 169
calls pfExport, 169
defun, 169
npExpress, 177
calledby npReturn, 176
calledby npStatement, 168
calls npExpress1, 177
calls npIterators, 177
calls npPop1, 177
calls npPop2, 177
calls npPush, 177
calls pfCollect, 177
calls pfListOf, 177
defun, 177
npExpress1, 177
calledby npExpress, 177
calls npADD, 177
calls npConditionalStatement, 177
defun, 177
npFirstTok, 141
calledby npNext, 143
calledby npParse, 139
calledby npRecoverTrap, 188
calledby npRestore, 150
calls tokConstruct, 141
calls tokPart, 141
calls tokPosn, 141
uses $inputStream, 141
uses $stok, 141
uses $ttok, 141
defun, 141
npFix, 164
calledby npPrimary1, 162
calls npDef, 164
calls npEqKey, 164
calls npPop1, 164
calls npPush, 164
calls pfFix, 164
defun, 164
npForIn, 175
calledby npIterators, 173
calledby npIterator, 174
calls npBy, 175
calls npCompMissing, 175
calls npEqKey, 175
calls npPop1, 176
calls npPop2, 176
calls npPush, 176
calls npTrap, 175
calls npVariable, 175
calls pfForin, 176
defun, 175
npFree, 171
calledby npStatement, 168
calls npEqKey, 171
calls npLocalItemlist, 171
calls npPop1, 171
calls npPush, 171
calls npTrap, 171
calls pfFree, 171
defun, 171
npFromdom, 200
calledby npAmpersandFrom, 200
calledby npAtom1, 181
calledby npAtom2, 158
calledby npEncAp, 180
calledby npSegment, 198
calls npApplication, 201
calls npEqKey, 200
calls npFromdom1, 201
calls npPop1, 201
calls npPush, 201
calls npTrap, 201
calls pfFromDom, 201
defun, 200
npFromdom1, 201
calledby npFromdom1, 201
calledby npFromdom, 201
calls npApplication, 201
calls npEqKey, 201
calls npFromdom1, 201
calls npPop1, 201
calls npPush, 201

calls npTrap, 201
calls pfFromDom, 201
defun, 201
npGives, 146
calledby npAssignment, 215
calledby npDefinitionOrStatement, 145
calls npBackTrack, 146
calls npExit, 146
calls npLambda, 146
defun, 146
npId, 202
calledby npName, 202
calledby npSymbolVariable, 203
calls npNext, 202
calls npPush, 202
calls tokConstruct, 202
calls tokPosn, 203
uses $npTokToNames, 203
uses $stok, 203
uses $ttok, 203
defun, 202
npImport, 178
calledby npDefinitionItem, 165
calledby npStatement, 168
calls npAndOr, 178
calls npQualTypelist, 178
calls pfImport, 178
defun, 178
npInfGeneric, 205
calledby npLeftAssoc, 205
calledby npRightAssoc, 204
calledby npTerm, 199
calls npDDInfKey, 205
calls npEqKey, 205
defun, 205
npInfixOp, 159
calledby npInfixOperator, 158
calls npPushId, 159
uses $stok, 159
uses $ttok, 159
defun, 159
npInfixOperator, 158
calledby npAtom2, 157
calledby npSignatureDefinee, 154
calls npEqKey, 158
calls npInfixOp, 158
calls npPop1, 158
calls npPush, 158
calls npRestore, 158
calls npState, 158
calls pfSymb, 158
calls tokConstruct, 158
calls tokPart, 158
calls tokPosn, 158
uses $stok, 158
defun, 158
npInfKey, 206
calledby npDDInfKey, 206
calls npPushId, 206
uses $stok, 206
uses $ttok, 206
defun, 206
npInline, 172
calledby npStatement, 168
calls npAndOr, 172
calls npQualTypelist, 172
calls pfInline, 172
defun, 172
npInterval, 197
calledby npBy, 197
calls npArith, 197
calls npEqPeek, 197
calls npPop1, 197
calls npPop2, 197
calls npPush, 197
calls npSegment, 197
calls pfApplication, 197
calls pfInfApplication, 197
defun, 197
npItem, 140
calledby npParse, 139
calls npEqKey, 140
calls npItem1, 140
calls npPop1, 140
calls npPush, 140
calls npQualDef, 140
calls pfEnSequence, 140
calls pfNovalue, 140
defun, 140
npItem1, 140
calledby npItem1, 140
calledby npItem, 140

calls npEqKey, 140
calls npItem1, 140
calls npPop1, 141
calls npQualDef, 140
defun, 140
npIterate, 172
calledby npStatement, 168
calls npEqKey, 172
calls npPush, 172
calls pfIterate, 172
calls pfNothing, 172
defun, 172
npIterator, 174
calledby npIterators, 173
calls npForIn, 174
calls npSuchThat, 174
calls npWhile, 174
defun, 174
npIterators, 173
calledby npExpress, 177
calledby npIterators, 174
calledby npLoop, 173
calls npForIn, 173
calls npIterators, 174
calls npIterator, 173
calls npPop1, 173
calls npPop2, 173
calls npPush, 173
calls npWhile, 173
calls npZeroOrMore, 173
defun, 173
npLambda, 146
calledby npGives, 146
calledby npLambda, 147
calls npDefinitionOrStatement, 147
calls npEqKey, 147
calls npLambda, 147
calls npPop1, 147
calls npPop2, 147
calls npPush, 147
calls npTrap, 147
calls npType, 147
calls npVariable, 146
calls pfLam, 147
calls pfReturnTyped, 147
defun, 146
npLeftAssoc, 204
calledby npArith, 198
calledby npBy, 197
calledby npDiscrim, 195
calledby npDisjand, 195
calledby npLogical, 195
calledby npMatch, 148
calledby npProduct, 200
calledby npRelation, 196
calledby npRemainder, 199
calledby npSuch, 148
calledby npSum, 199
calls npInfGeneric, 205
calls npPop1, 205
calls npPop2, 205
calls npPush, 205
calls pfApplication, 205
calls pfInfApplication, 205
defun, 204
npLet, 164
calledby npPrimary1, 162
calls npDefinitionOrStatement, 164
calls npLetQualified, 164
defun, 164
npLetQualified, 164
calledby npLet, 164
calledby npQualified, 145
calls npCompMissing, 164
calls npDefinition, 164
calls npEqKey, 164
calls npPop1, 165
calls npPop2, 165
calls npPush, 165
calls npTrap, 164
calls pfWhere, 165
defun, 164
nplisp, 468
calledby handleNoParseCommands, 466
calledby intnplisp, 36
uses $ans, 468
defun, 468
npList, 153
calledby npListing, 153
calls npEqKey, 153
calls npPop1, 153
calls npPop2, 153

calls npPop3, 153
calls npPush, 153
calls npTrap, 153
uses $stack, 153
defun, 153
npListAndRecover, 187
calledby npPPg, 208
calledby npPileDefinitionlist, 187
calls npEqKey, 187
calls npEqPeek, 187
calls npNext, 187
calls npPop1, 187
calls npPush, 187
calls npRecoverTrap, 187
calls syGeneralErrorHere, 187
uses $inputStream, 187
uses $stack, 187
catches, 187
defun, 187
npListing, 153
calledby npSDefaultItem, 167
calledby npSLocalItem, 170
calledby npSQualTypelist, 179
calledby npSigItemlist, 152
calledby npTypeVariablelist, 155
calledby npVariablelist, 211
calls npList, 153
calls pfListOf, 153
defun, 153
npListofFun, 219
calledby npSemiListing, 191
calledby npTuple, 144
calls npPop1, 219
calls npPop2, 219
calls npPop3, 219
calls npPush, 219
calls npTrap, 219
uses $stack, 219
defun, 219
npLocal, 171
calledby npStatement, 168
calls npEqKey, 171
calls npLocalItemlist, 171
calls npPop1, 171
calls npPush, 171
calls npTrap, 171
calls pfLocal, 171
defun, 171
npLocalDecl, 170
calledby npLocalItem, 170
calls npEqKey, 170
calls npPop1, 170
calls npPop2, 170
calls npPush, 170
calls npTrap, 170
calls npType, 170
calls pfNothing, 171
calls pfParts, 170
calls pfSpread, 170
defun, 170
npLocalItem, 170
calledby npSLocalItem, 170
calls npLocalDecl, 170
calls npTypeVariable, 170
defun, 170
npLocalItemlist, 169
calledby npExport, 169
calledby npFree, 171
calledby npLocal, 171
calls npPC, 169
calls npPop1, 169
calls npPush, 169
calls npSLocalItem, 169
calls pfUnSequence, 169
defun, 169
npLogical, 195
calledby npConditional, 193
calledby npSuchThat, 174
calledby npSuch, 148
calledby npWhile, 175
calls npDisjand, 195
calls npLeftAssoc, 195
defun, 195
npLoop, 173
calledby npStatement, 168
calls npAssign, 173
calls npCompMissing, 173
calls npEqKey, 173
calls npIterators, 173
calls npPop1, 173
calls npPop2, 173
calls npPush, 173

calls npTrap, 173
calls pfLoop1, 173
calls pfLp, 173
defun, 173
npMacro, 162
calledby npDefinitionItem, 165
calledby npPrimary1, 162
calls npMdef, 162
calls npPP, 162
defun, 162
npMatch, 148
calledby npDef, 185
calledby npType, 147
calls npLeftAssoc, 148
calls npSuch, 148
defun, 148
npMDEF, 163
calledby npAssign, 214
calls npBackTrack, 163
calls npMDEFinition, 163
calls npStatement, 163
defun, 163
npMdef, 162
calledby npMDEFinition, 163
calledby npMacro, 162
calls npDefTail, 163
calls npPop1, 163
calls npPush, 163
calls npQuiver, 162
calls npTrap, 163
calls pfCheckMacroOut, 162
calls pfMacro, 163
calls pfPushMacroBody, 163
defun, 162
npMDEFinition, 163
calledby npMDEF, 163
calls npMdef, 163
calls npPP, 163
defun, 163
npMissing, 149
calledby npCompMissing, 149
calledby npConditional, 193
calledby npMissingMate, 213
calledby npPileBracketed, 186
calls ncSoftError, 149
calls pname, 149
calls tokPosn, 149
uses $stok, 149
defun, 149
throws, 149
npMissingMate, 213
calledby npEnclosed, 209
calledby npParenthesize, 213
calls ncSoftError, 213
calls npMissing, 213
calls tokPosn, 213
defun, 213
npMoveTo, 189
calledby npMoveTo, 189
calledby npRecoverTrap, 188
calls npEqKey, 189
calls npEqPeek, 189
calls npMoveTo, 189
calls npNext, 189
uses $inputStream, 189
defun, 189
npName, 202
calledby npAmpersand, 202
calledby npAtom1, 181
calledby npReturn, 176
calledby npSignatureDefinee, 154
calledby npVariableName, 211
calls npId, 202
calls npSymbolVariable, 202
defun, 202
npNext, 143
calledby npConstTok, 182
calledby npDollar, 182
calledby npEqKey, 143
calledby npId, 202
calledby npListAndRecover, 187
calledby npMoveTo, 189
calledby npPrefixColon, 159
calledby npPushId, 207
calls npFirstTok, 143
uses $inputStream, 143
defun, 143
npNull, 331
calledby eqpileTree, 337
calledby insertpile, 333
calledby nextline, 110
calledby pileForests, 336

calledby pilePlusComments, 334
calledby pileTree, 335
calls StreamNull, 331
defun, 331
npParened, 183
calledby npBracketed, 183
calledby npPP, 207
calls npEnclosed, 183
calls pfParen, 183
defun, 183
npParenthesize, 213
calledby npParenthesized, 212
calls npEqKey, 213
calls npMissingMate, 213
calls npPush, 213
uses $stok, 213
defun, 213
npParenthesized, 212
calledby npPDefinition, 181
calledby npTypeVariable, 154
calledby npVariable, 211
calls npParenthesize, 212
defun, 212
npParse, 139
calledby intloopEchoParse, 69
calledby ncloopParse, 38
calls ncSoftError, 139
calls npFirstTok, 139
calls npItem, 139
calls pfDocument, 139
calls pfListOf, 139
calls pfWrong, 139
calls tokPosn, 139
uses $inputStream, 139
uses $stack, 139
uses $stok, 139
uses $ttok, 139
catches, 139
defun, 139
npPC
calledby npDefaultItemlist, 166
calledby npLocalItemlist, 169
calledby npQualTypelist, 178
npPDefinition, 181
calledby npAtom1, 181
calledby npBDefinition, 183
calls npDefinitionlist, 181
calls npParenthesized, 181
calls npPop1, 181
calls npPush, 181
calls pfEnSequence, 181
defun, 181
npPileBracketed, 186
calledby npBPileDefinition, 186
calledby npPP, 208
calls npEqKey, 186
calls npMissing, 186
calls npPop1, 186
calls npPush, 186
calls pfNothing, 186
calls pfPile, 186
defun, 186
npPileDefinitionlist, 187
calledby npBPileDefinition, 186
calls npDefinitionlist, 187
calls npListAndRecover, 187
calls npPop1, 187
calls npPush, 187
calls pfAppend, 187
defun, 187
npPileExit, 214
calledby npExit, 214
calls npAssign, 214
calls npEqKey, 214
calls npPop1, 214
calls npPop2, 214
calls npPush, 214
calls npStatement, 214
calls pfExit, 214
defun, 214
npPop1, 142
calledby npADD, 156
calledby npAdd, 157
calledby npAndOr, 180
calledby npApplication2, 161
calledby npApplication, 160
calledby npAssignVariable, 215
calledby npAssignment, 215
calledby npBPileDefinition, 186
calledby npCategoryL, 150
calledby npConstTok, 182
calledby npDDInfKey, 206

calledby npDecl, 212
calledby npDefaultDecl, 168
calledby npDefaultItemlist, 167
calledby npDefaultValue, 193
calledby npDefinition, 165
calledby npDef, 185
calledby npElse, 194
calledby npEnclosed, 209
calledby npEncl, 180
calledby npExport, 169
calledby npExpress, 177
calledby npFix, 164
calledby npForIn, 176
calledby npFree, 171
calledby npFromdom1, 201
calledby npFromdom, 201
calledby npInfixOperator, 158
calledby npInterval, 197
calledby npItem1, 141
calledby npItem, 140
calledby npIterators, 173
calledby npLambda, 147
calledby npLeftAssoc, 205
calledby npLetQualified, 165
calledby npListAndRecover, 187
calledby npListofFun, 219
calledby npList, 153
calledby npLocalDecl, 170
calledby npLocalItemlist, 169
calledby npLocal, 171
calledby npLoop, 173
calledby npMdef, 163
calledby npPDefinition, 181
calledby npPPff, 208
calledby npPPg, 208
calledby npPP, 208
calledby npPileBracketed, 186
calledby npPileDefinitionlist, 187
calledby npPileExit, 214
calledby npQualDef, 143
calledby npQualTypelist, 178
calledby npQualType, 179
calledby npQualified, 145
calledby npReturn, 176
calledby npRightAssoc, 204
calledby npSCategory, 151
calledby npSDefaultItem, 167
calledby npSLocalItem, 170
calledby npSQualTypelist, 179
calledby npSelector, 161
calledby npSigDecl, 155
calledby npSigItemlist, 152
calledby npSignature, 152
calledby npSingleRule, 192
calledby npSymbolVariable, 203
calledby npSynthetic, 196
calledby npTerm, 199
calledby npTypeVariable, 154
calledby npTypedForm1, 216
calledby npTypedForm, 218
calledby npType, 147
calledby npTyping, 166
calledby npVariableName, 212
calledby npVariable, 211
calledby npWConditional, 193
calledby npWith, 148
calledby npZeroOrMore, 175
uses $stack, 142
defun, 142
npPop2, 142
calledby npAdd, 157
calledby npApplication2, 161
calledby npApplication, 160
calledby npAssignment, 215
calledby npDecl, 212
calledby npDefaultDecl, 168
calledby npElse, 194
calledby npEncl, 180
calledby npExpress, 177
calledby npForIn, 176
calledby npInterval, 197
calledby npIterators, 173
calledby npLambda, 147
calledby npLeftAssoc, 205
calledby npLetQualified, 165
calledby npListofFun, 219
calledby npList, 153
calledby npLocalDecl, 170
calledby npLoop, 173
calledby npPileExit, 214
calledby npReturn, 176
calledby npRightAssoc, 204

calledby npSelector, 161
calledby npSigDecl, 155
calledby npSingleRule, 192
calledby npSynthetic, 196
calledby npTerm, 199
calledby npTypedForm1, 216
calledby npTypedForm, 218
calledby npWith, 148
calledby npZeroOrMore, 175
uses $stack, 142
defun, 142
npPop3, 142
calledby npElse, 194
calledby npListofFun, 219
calledby npList, 153
uses $stack, 142
defun, 142
npPower, 200
calledby npProduct, 200
calls npColon, 200
calls npRightAssoc, 200
defun, 200
npPP, 207
calledby npCategory, 151
calledby npDefinition, 165
calledby npDefn, 185
calledby npMDEFinition, 163
calledby npMacro, 162
calledby npRule, 191
calls npPPf, 207
calls npPPg, 208
calls npParened, 207
calls npPileBracketed, 208
calls npPop1, 208
calls npPush, 208
calls pfEnSequence, 208
uses npPParg, 208
defun, 207
npPParg, 202, 207
usedby npPP, 208
defvar, 202, 207
npPPf, 209
calledby npPPg, 208
calledby npPP, 207
calls npPPff, 209
calls npSemiListing, 209
defun, 209
npPPff, 208
calledby npPPf, 209
calls npPop1, 208
calls npPush, 208
uses $npPParg, 208
defun, 208
npPPg, 208
calledby npPP, 208
calls npListAndRecover, 208
calls npPPf, 208
calls npPop1, 208
calls npPush, 208
calls pfAppend, 208
defun, 208
npPrefixColon, 159
calledby npAtom2, 158
calledby npSignatureDefinee, 154
calls npEqPeek, 159
calls npNext, 159
calls npPush, 159
calls tokConstruct, 159
calls tokPosn, 159
uses $stok, 159
defun, 159
npPretend, 217
calledby npTypeStyle, 217
calls npTypedForm, 217
calls pfPretend, 217
defun, 217
npPrimary, 155
calledby npApplication, 160
calledby npSCategory, 151
calledby npSelector, 161
calls npPrimary1, 155
calls npPrimary2, 156
defun, 155
npPrimary1, 162
calledby npApplication2, 161
calledby npConstTok, 182
calledby npPrimary, 155
calls npAtom1, 162
calls npBPileDefinition, 162
calls npDefn, 162
calls npEncAp, 162
calls npFix, 162

calls npLet, 162
calls npMacro, 162
calls npRule, 162
defun, 162
npPrimary2, 156
calledby npPrimary, 156
calls npAdd, 156
calls npAtom2, 156
calls npEncAp, 156
calls npWith, 156
calls pfNothing, 156
defun, 156
npProcessSynonym, 469
calledby npsynonym, 469
calls printSynonyms, 469
calls processSynonymLine, 469
calls putalist, 469
calls terminateSystemCommand, 469
uses $CommandSynonymAlist, 469
defun, 469
npProduct, 200
calledby npRemainder, 199
calls npLeftAssoc, 200
calls npPower, 200
defun, 200
npPush, 141
calledby npADD, 156
calledby npAdd, 157
calledby npAndOr, 180
calledby npApplication2, 161
calledby npApplication, 160
calledby npAssignVariable, 215
calledby npAssignment, 215
calledby npBPileDefinition, 186
calledby npBreak, 172
calledby npCategoryL, 150
calledby npConstTok, 182
calledby npDDInfKey, 206
calledby npDecl, 212
calledby npDefaultDecl, 168
calledby npDefaultItemlist, 167
calledby npDefaultValue, 193
calledby npDefinition, 165
calledby npDef, 185
calledby npDollar, 181
calledby npElse, 194
calledby npEnclosed, 209
calledby npEncl, 180
calledby npExport, 169
calledby npExpress, 177
calledby npFix, 164
calledby npForIn, 176
calledby npFree, 171
calledby npFromdom1, 201
calledby npFromdom, 201
calledby npId, 202
calledby npInfixOperator, 158
calledby npInterval, 197
calledby npItem, 140
calledby npIterate, 172
calledby npIterators, 173
calledby npLambda, 147
calledby npLeftAssoc, 205
calledby npLetQualified, 165
calledby npListAndRecover, 187
calledby npListofFun, 219
calledby npList, 153
calledby npLocalDecl, 170
calledby npLocalItemlist, 169
calledby npLocal, 171
calledby npLoop, 173
calledby npMdef, 163
calledby npPDefinition, 181
calledby npPPff, 208
calledby npPPg, 208
calledby npPP, 208
calledby npParenthesize, 213
calledby npPileBracketed, 186
calledby npPileDefinitionlist, 187
calledby npPileExit, 214
calledby npPrefixColon, 159
calledby npQualDef, 143
calledby npQualTypelist, 178
calledby npQualType, 179
calledby npQualified, 145
calledby npRecoverTrap, 188
calledby npReturn, 176
calledby npRightAssoc, 204
calledby npSCategory, 151
calledby npSDefaultItem, 167
calledby npSLocalItem, 170
calledby npSQualTypelist, 179

calledby npSelector, 161
calledby npSigDecl, 155
calledby npSigItemlist, 152
calledby npSignature, 152
calledby npSingleRule, 192
calledby npSymbolVariable, 203
calledby npSynthetic, 196
calledby npTerm, 199
calledby npTypeVariable, 154
calledby npTypedForm1, 216
calledby npTypedForm, 218
calledby npType, 147
calledby npTyping, 166
calledby npVariableName, 212
calledby npVariable, 211
calledby npWConditional, 193
calledby npWith, 148
calledby npZeroOrMore, 175
uses $stack, 141
defun, 141
npPushId, 207
calledby npInfKey, 206
calledby npInfixOp, 159
calledby npSegment, 198
calls npNext, 207
calls tokConstruct, 207
calls tokPosn, 207
uses $stack, 207
uses $stok, 207
uses $ttok, 207
defun, 207
npQualDef, 143
calledby npDefinitionlist, 191
calledby npItem1, 140
calledby npItem, 140
calls npComma, 143
calls npPop1, 143
calls npPush, 143
defun, 143
npQualified, 145
calledby npQualifiedDefinition, 145
calls npDefinition, 145
calls npEqKey, 145
calls npLetQualified, 145
calls npPop1, 145
calls npPush, 145
calls npTrap, 145
calls pfWhere, 145
defun, 145
npQualifiedDefinition, 145
calledby npComma, 144
calledby npConditionalStatement, 178
calls npDefinitionOrStatement, 145
calls npQualified, 145
defun, 145
npQualType, 179
calledby npSQualTypelist, 179
calls npPop1, 179
calls npPush, 179
calls npType, 179
calls pfNothing, 179
calls pfQualType, 179
defun, 179
npQualTypelist, 178
calledby npImport, 178
calledby npInline, 172
calls npPC, 178
calls npPop1, 178
calls npPush, 178
calls npSQualTypelist, 178
calls pfUnSequence, 178
defun, 178
npQuiver, 196
calledby npDiscrim, 195
calledby npMdef, 162
calledby npSingleRule, 192
calls npRelation, 196
calls npRightAssoc, 196
defun, 196
npRecoverTrap, 188
calledby npListAndRecover, 187
calls npFirstTok, 188
calls npMoveTo, 188
calls npPush, 188
calls pfDocument, 188
calls pfListOf, 188
calls pfWrong, 188
calls syIgnoredFromTo, 188
calls tokPosn, 188
uses $stok, 188
defun, 188
npRelation, 196

calledby npQuiver, 196
calls npLeftAssoc, 196
calls npSynthetic, 196
defun, 196
npRemainder, 199
calledby npTerm, 199
calls npLeftAssoc, 199
calls npProduct, 199
defun, 199
npRestore, 150
calledby npAdd, 157
calledby npBackTrack, 146
calledby npConstTok, 182
calledby npDDInfKey, 206
calledby npDefinitionItem, 165
calledby npElse, 194
calledby npInfixOperator, 158
calledby npRightAssoc, 204
calledby npSCategory, 151
calledby npSymbolVariable, 203
calledby npWith, 148
calls npFirstTok, 150
uses $inputStream, 150
uses $stack, 150
defun, 150
npRestrict, 218
calledby npTypeStyle, 217
calls npTypedForm, 218
calls pfRestrict, 218
defun, 218
npReturn, 176
calledby npStatement, 168
calls npEqKey, 176
calls npExpress, 176
calls npName, 176
calls npPop1, 176
calls npPop2, 176
calls npPush, 176
calls npTrap, 176
calls pfNothing, 176
calls pfReturnNoName, 176
calls pfReturn, 176
defun, 176
npRightAssoc, 204
calledby npPower, 200
calledby npQuiver, 196
calledby npRightAssoc, 204
calls npInfGeneric, 204
calls npPop1, 204
calls npPop2, 204
calls npPush, 204
calls npRestore, 204
calls npRightAssoc, 204
calls npState, 204
calls pfApplication, 204
calls pfInfApplication, 204
defun, 204
npRule, 191
calledby npPrimary1, 162
calls npEqKey, 191
calls npPP, 191
calls npSingleRule, 191
defun, 191
npSCategory, 151
calledby npCategory, 151
calls npApplication, 151
calls npCategoryL, 151
calls npDefaultValue, 151
calls npEqPeek, 151
calls npPop1, 151
calls npPrimary, 151
calls npPush, 151
calls npRestore, 151
calls npSignature, 151
calls npState, 151
calls npTrap, 151
calls npWConditional, 151
calls pfAttribute, 151
defun, 151
npSDefaultItem, 167
calledby npDefaultItemlist, 167
calls npDefaultItem, 167
calls npListing, 167
calls npPop1, 167
calls npPush, 167
calls pfAppend, 167
calls pfParts, 167
defun, 167
npSegment, 198
calledby npInterval, 197
calls npEqPeek, 198
calls npFromdom, 198

calls npPushId, 198
defun, 198
npSelector, 161
calls npEqKey, 161
calls npPop1, 161
calls npPop2, 161
calls npPrimary, 161
calls npPush, 161
calls npTrap, 161
calls pfApplication, 161
defun, 161
npSemiBackSet, 191
calledby npSemiListing, 191
calls npEqKey, 191
defun, 191
npSemiListing, 191
calledby npDefinitionlist, 191
calledby npPPf, 209
calls npListofFun, 191
calls npSemiBackSet, 191
calls pfAppend, 191
defun, 191
npSigDecl, 155
calledby npSigItem, 154
calls npEqKey, 155
calls npPop1, 155
calls npPop2, 155
calls npPush, 155
calls npTrap, 155
calls npType, 155
calls pfParts, 155
calls pfSpread, 155
defun, 155
npSigItem, 154
calledby npSigItemlist, 152
calls npSigDecl, 154
calls npTrap, 154
calls npTypeVariable, 154
defun, 154
npSigItemlist, 152
calledby npSignature, 152
calls npListing, 152
calls npPop1, 152
calls npPush, 152
calls npSigItem, 152
calls pfAppend, 152
calls pfListOf, 152
calls pfParts, 152
defun, 152
npSignature, 152
calledby npSCategory, 151
calls npPop1, 152
calls npPush, 152
calls npSigItemlist, 152
calls pfNothing, 152
calls pfWDec, 152
defun, 152
npSignatureDefinee, 154
calledby npTypeVariablelist, 155
calledby npTypeVariable, 154
calls npInfixOperator, 154
calls npName, 154
calls npPrefixColon, 154
defun, 154
npSingleRule, 192
calledby npRule, 191
calls npDefTail, 192
calls npPop1, 192
calls npPop2, 192
calls npPush, 192
calls npQuiver, 192
calls npTrap, 192
calls pfRule, 192
defun, 192
npSLocalItem, 170
calledby npLocalItemlist, 169
calls npListing, 170
calls npLocalItem, 170
calls npPop1, 170
calls npPush, 170
calls pfAppend, 170
calls pfParts, 170
defun, 170
npSQualTypelist, 179
calledby npQualTypelist, 178
calls npListing, 179
calls npPop1, 179
calls npPush, 179
calls npQualType, 179
calls pfParts, 179
defun, 179
npState, 210

calledby npAdd, 157
calledby npBackTrack, 146
calledby npConstTok, 182
calledby npDDInfKey, 206
calledby npDefinitionItem, 165
calledby npElse, 194
calledby npInfixOperator, 158
calledby npRightAssoc, 204
calledby npSCategory, 151
calledby npSymbolVariable, 203
calledby npWith, 148
uses $inputStream, 210
uses $stack, 210
defun, 210
npStatement, 168
calledby npDefinitionItem, 165
calledby npMDEF, 163
calledby npPileExit, 214
calledby npVoid, 177
calls npBreak, 168
calls npExport, 168
calls npExpress, 168
calls npFree, 168
calls npImport, 168
calls npInline, 168
calls npIterate, 168
calls npLocal, 168
calls npLoop, 168
calls npReturn, 168
calls npTyping, 168
calls npVoid, 168
defun, 168
npSuch, 148
calledby npMatch, 148
calls npLeftAssoc, 148
calls npLogical, 148
defun, 148
npSuchThat, 174
calledby npIterator, 174
calls npAndOr, 174
calls npLogical, 174
calls pfSuchthat, 174
defun, 174
npSum, 199
calledby npArith, 198
calls npLeftAssoc, 199
calls npTerm, 199
defun, 199
npSymbolVariable, 203
calledby npName, 202
calls npEqKey, 203
calls npId, 203
calls npPop1, 203
calls npPush, 203
calls npRestore, 203
calls npState, 203
calls tokConstruct, 203
calls tokPart, 203
calls tokPosn, 203
defun, 203
npsynonym, 469
calledby handleNoParseCommands, 466
calls npProcessSynonym, 469
defun, 469
npSynthetic, 196
calledby npRelation, 196
calls npAmpersandFrom, 196
calls npBy, 196
calls npPop1, 196
calls npPop2, 196
calls npPush, 196
calls pfApplication, 196
calls pfInfApplication, 196
defun, 196
npsystem, 468
calledby handleNoParseCommands, 466
calls sayKeyedMsg, 468
defun, 468
npTagged, 216
calledby npColon, 215
calls npTypedForm1, 216
calls pfTagged, 216
defun, 216
npTerm, 199
calledby npSum, 199
calls npInfGeneric, 199
calls npPop1, 199
calls npPop2, 199
calls npPush, 199
calls npRemainder, 199
calls pfApplication, 199
defun, 199

npTrap, 210
calledby npAdd, 157
calledby npAmpersand, 202
calledby npAndOr, 180
calledby npAssignment, 215
calledby npBackTrack, 146
calledby npConditional, 193
calledby npDecl, 212
calledby npDefaultDecl, 168
calledby npDefaultItem, 167
calledby npDefaultValue, 193
calledby npDefinitionItem, 166
calledby npDef, 185
calledby npElse, 194
calledby npExport, 169
calledby npForIn, 175
calledby npFree, 171
calledby npFromdom1, 201
calledby npFromdom, 201
calledby npLambda, 147
calledby npLetQualified, 164
calledby npListofFun, 219
calledby npList, 153
calledby npLocalDecl, 170
calledby npLocal, 171
calledby npLoop, 173
calledby npMdef, 163
calledby npQualified, 145
calledby npReturn, 176
calledby npSCategory, 151
calledby npSelector, 161
calledby npSigDecl, 155
calledby npSigItem, 154
calledby npSingleRule, 192
calledby npTypedForm1, 216
calledby npTypedForm, 218
calledby npTyping, 166
calledby npWith, 148
calls ncSoftError, 210
calls tokPosn, 210
uses $stok, 210
defun, 210
throws, 210
npTrapForm, 210
calledby pfCheckId, 239
calledby pfCheckItOut, 237
calledby pfCheckMacroOut, 238
calls ncSoftError, 210
calls pfSourceStok, 210
calls syGeneralErrorHere, 210
calls tokPosn, 210
defun, 210
throws, 210
npTuple, 144
calledby npComma, 144
calls npCommaBackSet, 144
calls npListofFun, 144
calls pfTupleListOf, 144
defun, 144
npType, 147
calledby npADD, 156
calledby npDecl, 212
calledby npDefaultDecl, 168
calledby npLambda, 147
calledby npLocalDecl, 170
calledby npQualType, 179
calledby npSigDecl, 155
calledby npTypedForm1, 216
calls npMatch, 147
calls npPop1, 147
calls npPush, 147
calls npWith, 147
defun, 147
npTypedForm, 218
calledby npCoerceTo, 218
calledby npColonQuery, 217
calledby npPretend, 217
calledby npRestrict, 218
calls npApplication, 218
calls npEqKey, 218
calls npPop1, 218
calls npPop2, 218
calls npPush, 218
calls npTrap, 218
defun, 218
npTypedForm1, 216
calledby npTagged, 216
calls npEqKey, 216
calls npPop1, 216
calls npPop2, 216
calls npPush, 216
calls npTrap, 216

calls npType, 216
defun, 216
npTypeStyle, 217
calledby npTypified, 216
calls npCoerceTo, 217
calls npColonQuery, 217
calls npPretend, 217
calls npRestrict, 217
defun, 217
npTypeVariable, 154
calledby npDefaultItem, 167
calledby npLocalItem, 170
calledby npSigItem, 154
calls npParenthesized, 154
calls npPop1, 154
calls npPush, 154
calls npSignatureDefinee, 154
calls npTypeVariablelist, 154
calls pfListOf, 154
defun, 154
npTypeVariablelist, 155
calledby npTypeVariable, 154
calls npListing, 155
calls npSignatureDefinee, 155
defun, 155
npTypified, 216
calledby npColon, 215
calls npAnyNo, 216
calls npApplication, 216
calls npTypeStyle, 216
defun, 216
npTyping, 166
calledby npDefinitionItem, 165
calledby npStatement, 168
calls npDefaultItemlist, 166
calls npEqKey, 166
calls npPop1, 166
calls npPush, 166
calls npTrap, 166
calls pfTyping, 166
defun, 166
npVariable, 211
calledby npAdd, 157
calledby npForIn, 175
calledby npLambda, 146
calledby npWith, 148
calls npParenthesized, 211
calls npPop1, 211
calls npPush, 211
calls npVariableName, 211
calls npVariablelist, 211
calls pfListOf, 211
defun, 211
npVariablelist, 211
calledby npVariable, 211
calls npListing, 211
calls npVariableName, 211
defun, 211
npVariableName, 211
calledby npVariablelist, 211
calledby npVariable, 211
calls npDecl, 212
calls npName, 211
calls npPop1, 212
calls npPush, 212
calls pfNothing, 212
calls pfTyped, 212
defun, 211
npVoid, 177
calledby npStatement, 168
calls npAndOr, 177
calls npStatement, 177
calls pfNovalue, 177
defun, 177
npWConditional, 193
calledby npSCategory, 151
calls npConditional, 193
calls npPop1, 193
calls npPush, 193
calls pfTweakIf, 193
defun, 193
npWhile, 175
calledby npIterators, 173
calledby npIterator, 174
calls npAndOr, 175
calls npLogical, 175
calls pfWhile, 175
defun, 175
npWith, 148
calledby npPrimary2, 156
calledby npType, 147
calls npCategoryL, 148

calls npCompMissing, 148
calls npEqKey, 148
calls npEqPeek, 148
calls npPop1, 148
calls npPop2, 148
calls npPush, 148
calls npRestore, 148
calls npState, 148
calls npTrap, 148
calls npVariable, 148
calls pfNothing, 148
calls pfWith, 148
defun, 148
npZeroOrMore, 175
calledby npIterators, 173
calls npPop1, 175
calls npPop2, 175
calls npPush, 175
uses $stack, 175
defun, 175
nremove
calledby changeToNamedInterpreterFrame, 556
nreverse0
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
NRTevalDomain, 1064
calledby compiledLookup, 1061
calls evalDomain, 1064
calls eval, 1064
calls qcar, 1064
defun, 1064
nsubst
calledby ruleLhsTran, 320
calledby zeroOneTran, 68

obey
calledby copyright, 518
calledby editFile, 541
calledby newHelpSpad2Cmd, 568
calledby summary, 822
object2Identifier
calledby fetchKeyedMsg, 326
calledby fixObjectForPrinting, 452
calledby frameSpad2Cmd, 562
calledby pathnameTypeId, 1035
calledby readHiFi, 597
calledby saveHistory, 592
calledby selectOptionLC, 475
calledby setHistoryCore, 581
calledby writeHiFi, 598
object2String
calledby clearCmdExcept, 501
calledby displayMacro, 450
calledby displaySetOptionInformation, 647
calledby displaySetVariableSettings, 649
calledby getTraceOption, 844
calledby makePathname, 1036
calledby set1, 800
calledby setExposeAddGroup, 690
calledby setLinkerArgs, 720
calledby setNagHost, 746
calledby setOutputAlgebra, 754
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby setStreamsCalculate, 793
objMode
calledby displayType, 456
calledby displayValue, 455
calledby getAndEvalConstructorArgument, 948
calledby interpret2, 55
calledby isDomainValuedVariable, 949
calledby printTypeAndTimeNormal, 59
calledby processInteractive1, 52
calledby retract, 1049
calledby showInOut, 596
calledby transTraceItem, 853
objNew
calledby interpret1, 54
calledby interpret2, 55
calledby retract, 1049
objNewWrap
calledby coerceSpadArgs2E, 855
calledby coerceSpadFunValue2E, 858
calledby coerceTraceArgs2E, 854
calledby coerceTraceFunValue2E, 857

calledby getAndEvalConstructorArgument, 948
calledby printTypeAndTimeNormal, 59
calledby recordAndPrint, 56
objVal
calledby evaluateType, 906
calledby getAndEvalConstructorArgument, 948
calledby interpret2, 55
calledby retract, 1049
calledby transTraceItem, 853
objValUnwrap
calledby coerceSpadArgs2E, 855
calledby coerceSpadFunValue2E, 858
calledby coerceTraceArgs2E, 854
calledby coerceTraceFunValue2E, 857
calledby displayValue, 455
calledby evaluateType1, 908
calledby isDomainValuedVariable, 949
calledby processInteractive1, 52
calledby showInOut, 596
ofCategory, 417
calledby evalCategory, 949
calledby ofCategory, 417
calls hasCaty, 417
calls identp, 417
calls ofCategory, 417
local def $Subst, 417
local def $hope, 417
defun, 417
oldCompLookup, 1064
calledby basicLookup, 1061
calls lookupInDomainVector, 1064
local def $lookupDefaults, 1064
defun, 1064
oldHistFileName, 576
calledby initHist, 577
calls makeHistFileName, 576
uses $oldHistoryFileName, 576
defun, 576
om-bindTCP, 1125
defun, 1125
om-closeConn, 1124
defun, 1124
om-closeDev, 1124
defun, 1124
om-connectTCP, 1126
defun, 1126
om-getApp, 1127
defun, 1127
om-getAtp, 1128
defun, 1128
om-getAttr, 1128
defun, 1128
om-getBind, 1128
defun, 1128
om-getBVar, 1128
defun, 1128
om-getByteArray, 1129
defun, 1129
om-getConnInDev, 1125
defun, 1125
om-getConnOutDev, 1125
defun, 1125
om-getEndApp, 1129
defun, 1129
om-getEndAtp, 1129
defun, 1129
om-getEndAttr, 1129
defun, 1129
om-getEndBind, 1130
defun, 1130
om-getEndBVar, 1130
defun, 1130
om-getEndError, 1130
defun, 1130
om-getEndObject, 1130
defun, 1130
om-getError, 1131
defun, 1131
om-getFloat, 1131
defun, 1131
om-getInt, 1131
defun, 1131
om-getObject, 1131
defun, 1131
om-getString, 1132
defun, 1132
om-getSymbol, 1132
defun, 1132
om-getType, 1132
defun, 1132

om-getVar, 1132
defun, 1132
om-listCDs, 1122
defun, 1122
om-listSymbols, 1122
defun, 1122
om-makeConn, 1124
defun, 1124
om-openFileDev, 1123
defun, 1123
om-openStringDev, 1124
defun, 1124
om-putApp, 1133
defun, 1133
om-putAtp, 1133
defun, 1133
om-putAttr, 1133
defun, 1133
om-putBind, 1133
defun, 1133
om-putBVar, 1134
defun, 1134
om-putByteArray, 1134
defun, 1134
om-putEndApp, 1134
defun, 1134
om-putEndAtp, 1134
defun, 1134
om-putEndAttr, 1135
defun, 1135
om-putEndBind, 1135
defun, 1135
om-putEndBVar, 1135
defun, 1135
om-putEndError, 1135
defun, 1135
om-putEndObject, 1136
defun, 1136
om-putError, 1136
defun, 1136
om-putFloat, 1136
defun, 1136
om-putInt, 1136
defun, 1136
om-putObject, 1137
defun, 1137
om-putString, 1137
defun, 1137
om-putSymbol, 1137
defun, 1137
om-putVar, 1137
defun, 1137
om-Read, 1121
defun, 1121
om-setDevEncoding, 1123
defun, 1123
om-stringPtrToString, 1138
defun, 1138
om-stringToStringPtr, 1138
defun, 1138
om-supportsCD, 1122
defun, 1122
om-supportsSymbol, 1122
defun, 1122
openOutputLibrary, 658
calledby setOutputLibrary, 656
calls dropInputLibrary, 658
uses input-libraries, 658
uses output-library, 658
defun, 658
openserver, 977
calledby restart, 17
defun, 977
operationOpen, 998
calls unsqueeze, 998
uses *operation-hash*, 998
uses *operation-stream*, 998
uses *operation-stream-stamp*, 998
uses $spadroot, 998
defun, 998
operationopen
calledby resethashtables, 991
calledby restart0, 19
opIsHasCat
calledby basicLookup, 1061
opOf
calledby abbreviationsSpad2Cmd, 479
calledby displaySpad2Cmd, 531
calledby domainToGenvar, 851
calledby hasCaty, 418
calledby isDomainOrPackage, 865
calledby reportOperations, 808

calledby spadTrace, 866
optionError, 445
calledby readSpad2Cmd, 638
calls commandErrorMessage, 445
defun, 445
optionUserLevelError, 446
calls userLevelErrorMessage, 446
defun, 446
opTran, 321
calledby pf2Sex1, 298
calledby pfApplication2Sex, 303
uses $dotdot, 321
defun, 321
orderBySlotNumber, 883
calls assocright, 883
calls exit, 883
calls orderList, 883
calls seq, 883
defun, 883
orderList
calledby orderBySlotNumber, 883
origin
calledby inclmsgIfSyntax, 97
calledby inclmsgPrematureEOF, 88
calledby inclmsgPrematureFin, 95
out-stream, 952
usedby spad-long-error, 959
defvar, 952
output
calledby recordAndPrint, 56
output-library, 657
usedby openOutputLibrary, 658
defvar, 657
outputFormat
calledby displayValue, 455

parallel, 397
syntax, 397
parseAndInterpret, 48
calledby executeQuietCommand, 47
calledby serverReadLine, 44
uses $InteractiveFrame, 48
uses $InteractiveMode, 48
uses $boot, 48
uses $e, 48
uses $spad, 48
defun, 48
parseFromString, 48
calledby parseSystemCmd, 465
calls StreamNull, 48
calls incString, 48
calls lineoftoks, 48
calls macroExpanded, 48
calls ncloopParse, 48
calls next, 48
calls pf2Sex, 48
defun, 48
parseSystemCmd, 465
calledby handleParsedSystemCommands, 464
calls dumbTokenize, 465
calls parseFromString, 465
calls stripSpaces, 465
calls tokTran, 465
defun, 465
pathname, 1036
calledby deleteFile, 1037
calledby editFile, 541
calledby editSpad2Cmd, 540
calledby makePathname, 1036
calledby namestring, 1034
calledby pathnameDirectory, 1035
calledby pathnameName, 1034
calledby pathnameType, 1034
calledby pathname, 1036
calledby readSpad2Cmd, 638
calledby reportOpsFromLisplib1, 809
calledby reportOpsFromUnitDirectly1, 817
calledby setExposeAddGroup, 691
calledby setExpose, 689
calledby updateSourceFiles, 542
calledby workfilesSpad2Cmd, 939
calls make-filename, 1036
calls pathname, 1036
defun, 1036
pathnameDirectory, 1035
calledby editSpad2Cmd, 540
calledby loadLib, 1029
calledby mergePathnames, 1035
calledby setOutputAlgebra, 754
calledby setOutputFormula, 782
calledby setOutputFortran, 761

calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calls pathname, 1035
defun, 1035
pathnameName, 1034
calledby editSpad2Cmd, 540
calledby mergePathnames, 1035
calledby readSpad2Cmd, 638
calledby setOutputAlgebra, 754
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby updateSourceFiles, 542
calls pathname, 1034
defun, 1034
PathnameString
calledby pfname, 91
pathnameType, 1034
calledby editSpad2Cmd, 540
calledby mergePathnames, 1035
calledby pathnameTypeId, 1035
calledby setOutputAlgebra, 754
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby updateSourceFiles, 542
calls pathname, 1034
defun, 1034
pathnameTypeId, 1035
calledby readSpad2Cmd, 638
calledby updateSourceFiles, 542
calls object2Identifier, 1035
calls pathnameType, 1035
calls upcase, 1035
defun, 1035
patternVarsOf, 319
calledby ruleLhsTran, 320
calledby rulePredicateTran, 317
calls patternVarsOf1, 319
defun, 319
patternVarsOf1, 319
calledby patternVarsOf1, 319
calledby patternVarsOf, 319
calls patternVarsOf1, 319
defun, 319
pcounters, 850
calledby trace1, 838
calls bright, 850
calls concat, 850
calls sayBrightly, 850
uses /countlist, 850
defun, 850
peekTimedName
calledby interpretTopLevel, 53
pf0ApplicationArgs, 235
calledby macApplication, 221
calls pf0FlattenSyntacticTuple, 235
calls pfApplicationArg, 235
defun, 235
pf0AssignLhsItems, 254
calledby pf2Sex1, 299
calls pfAssignLhsItems, 254
calls pfParts, 254
defun, 254
pf0DefinitionLhsItems, 260
calledby pfDefinition2Sex, 313
calls pfDefinitionLhsItems, 260
calls pfParts, 260
defun, 260
pf0FlattenSyntacticTuple, 235
calledby pf0ApplicationArgs, 235
calledby pf0FlattenSyntacticTuple, 235
calls pf0FlattenSyntacticTuple, 235
calls pf0TupleParts, 235
calls pfTuple?, 235
defun, 235
pf0ForinLhs, 265
calledby pf2Sex1, 298
calls pfForinLhs, 265
calls pfParts, 265
defun, 265
pf0FreeItems, 264
calledby pf2Sex1, 299
calls pfFreeItems, 264

calls pfParts, 264
defun, 264
pf0LambdaArgs, 272
calledby macLambdaParameterHandling, 229
calledby pfLambdaTran, 314
calls pfLambdaArgs, 272
calls pfParts, 272
defun, 272
pf0LocalItems, 273
calledby pf2Sex1, 299
calls pfLocalItems, 273
calls pfParts, 273
defun, 273
pf0LoopIterators, 274
calledby pf0LoopIterators, 274
calledby pf2Sex1, 298
calls pf0LoopIterators, 274
calls pfParts, 274
defun, 274
pf0MLambdaArgs, 276
calledby mac0MLambdaApply, 221
calledby macLambdaParameterHandling, 229
calls pfParts, 276
defun, 276
pf0SequenceArgs, 285
calledby pfSequence2Sex, 308
calledby pfUnSequence, 291
calls pfParts, 285
calls pfSequenceArgs, 285
defun, 285
pf0TupleParts, 291
calledby pf0FlattenSyntacticTuple, 235
calledby pf2Sex1, 298
calledby pfApplication2Sex, 304
calledby pfCheckArg, 239
calledby pfCheckItOut, 237
calledby pfCollectVariable1, 240
calledby pfSexpr,strip, 248
calledby pfSuchThat2Sex, 305
calledby pfTransformArg, 242
calls pfParts, 291
calls pfTupleParts, 291
defun, 291
pf0WhereContext, 293
calledby pf2Sex1, 299
calls pfParts, 293
calls pfWhereContext, 293
defun, 293
pf2Sex, 297
calledby intInterpretPform, 67
calledby parseFromString, 48
calledby pfApplication2Sex, 304
calledby pfSuchThat2Sex, 305
calls pf2Sex1, 297
uses $QuietCommand, 297
uses $insideApplication, 297
uses $insideRule, 297
uses $insideSEQ, 297
defun, 297
pf2Sex1, 298
calledby loopIters2Sex, 310
calledby pf2Sex1, 298
calledby pf2Sex, 297
calledby pfApplication2Sex, 304
calledby pfCollect2Sex, 312
calledby pfDefinition2Sex, 313
calledby pfLambdaTran, 314
calledby pfLhsRule2Sex, 316
calledby pfOp2Sex, 306
calledby pfRhsRule2Sex, 317
calledby pfSequence2Sex, 308
calledby pfSuchThat2Sex, 305
calls keyedSystemError, 299
calls loopIters2Sex, 298
calls opTran, 298
calls pf0AssignLhsItems, 299
calls pf0ForinLhs, 298
calls pf0FreeItems, 299
calls pf0LocalItems, 299
calls pf0LoopIterators, 298
calls pf0TupleParts, 298
calls pf0WhereContext, 299
calls pf2Sex1, 298
calls pfAbSynOp, 300
calls pfAnd?, 299
calls pfAndLeft, 299
calls pfAndRight, 299
calls pfApplication2Sex, 298
calls pfApplication?, 298
calls pfAssign?, 299

calls pfAssignRhs, 299
calls pfBreak?, 299
calls pfBreakFrom, 299
calls pfCoerceto?, 298
calls pfCoercetoExpr, 298
calls pfCoercetoType, 298
calls pfCollect2Sex, 298
calls pfCollect?, 298
calls pfDefinition2Sex, 299
calls pfDefinition?, 299
calls pfDo?, 299
calls pfDoBody, 299
calls pfExit?, 298
calls pfExitCond, 298
calls pfExitExpr, 298
calls pfForin?, 298
calls pfForinWhole, 298
calls pfFree?, 299
calls pfFromdom?, 298
calls pfFromdomDomain, 298
calls pfFromdomWhat, 298
calls pfIdSymbol, 298
calls pfIf?, 298
calls pfIfCond, 298
calls pfIfElse, 298
calls pfIfThen, 298
calls pfIterate?, 299
calls pfLambda2Sex, 299
calls pfLambda?, 299
calls pfLiteral2Sex, 298
calls pfLiteral?, 298
calls pfLocal?, 299
calls pfLoop?, 298
calls pfMLambda?, 299
calls pfMacro?, 299
calls pfNot?, 299
calls pfNotArg, 299
calls pfNothing?, 298
calls pfNovalue?, 299
calls pfNovalueExpr, 299
calls pfOr?, 299
calls pfOrLeft, 299
calls pfOrRight, 299
calls pfPretend?, 298
calls pfPretendExpr, 298
calls pfPretendType, 298
calls pfRestrict?, 299
calls pfRestrictExpr, 299
calls pfRestrictType, 299
calls pfReturn?, 299
calls pfReturnExpr, 299
calls pfRule2Sex, 299
calls pfRule?, 299
calls pfSequence2Sex, 298
calls pfSequence?, 298
calls pfSuchthat?, 299
calls pfSuchthatCond, 299
calls pfSymbol?, 298
calls pfSymbolSymbol, 298
calls pfTagged?, 298
calls pfTaggedExpr, 298
calls pfTaggedTag, 298
calls pfTuple?, 298
calls pfTyped?, 299
calls pfTypedId, 299
calls pfTypedType, 299
calls pfWhere?, 299
calls pfWhereExpr, 299
calls pfWhile?, 298
calls pfWhileCond, 298
calls pfWrong?, 299
calls spadThrow, 299
calls tokPart, 300
uses $QuietCommand, 300
uses $insideRule, 300
uses $insideSEQ, 300
defun, 298
pf2sex1
calledby pfCollectArgTran, 315
pfAbSynOp, 410
calledby macLambdaParameterHandling, 229
calledby pf2Sex1, 300
calledby pfCopyWithPos, 234
calledby pfLeaf?, 245
calledby pfLiteral?, 246
calledby pfLiteralClass, 246
calledby pfMapParts, 235
calledby pfSexpr,strip, 248
calls ifcar, 410
defun, 410
pfAbSynOp?, 410

calledby intloopProcess, 64
calledby pfAnd?, 252
calledby pfApplication?, 253
calledby pfAssign?, 254
calledby pfBreak?, 256
calledby pfCoerceto?, 257
calledby pfCollect?, 258
calledby pfDefinition?, 259
calledby pfDo?, 260
calledby pfExit?, 261
calledby pfForin?, 264
calledby pfFree?, 263
calledby pfFromdom?, 266
calledby pfId?, 244
calledby pfIf?, 267
calledby pfIterate?, 269
calledby pfLambda?, 271
calledby pfLam, 270
calledby pfLocal?, 272
calledby pfLoop?, 274
calledby pfMLambda?, 276
calledby pfMacro?, 275
calledby pfNot?, 277
calledby pfNothing?, 243
calledby pfNovalue?, 278
calledby pfOr?, 278
calledby pfPretend?, 280
calledby pfRestrict?, 281
calledby pfReturn?, 282
calledby pfRule?, 284
calledby pfSequence?, 285
calledby pfSuchthat?, 286
calledby pfSymbol?, 250
calledby pfTagged?, 287
calledby pfTuple?, 290
calledby pfTyped?, 289
calledby pfWhere?, 292
calledby pfWhile?, 294
calledby pfWrong?, 295
calls eqcar, 410
defun, 410

pfAdd, 250
calledby npAdd, 157
calledby npDefaultValue, 193
calls pfNothing, 250
calls pfTree, 250
defun, 250

pfAnd, 251
calledby pfInfApplication, 269
calls pfTree, 251
defun, 251

pfAnd?, 252
calledby pf2Sex1, 299
calls pfAbSynOp?, 252
defun, 252

pfAndLeft, 252
calledby pf2Sex1, 299
defun, 252

pfAndRight, 253
calledby pf2Sex1, 299
defun, 253

pfAppend, 253
calledby npPPg, 208
calledby npPileDefinitionlist, 187
calledby npSDefaultItem, 167
calledby npSLocalItem, 170
calledby npSemiListing, 191
calledby npSigItemlist, 152
calledby pfUnSequence, 291
defun, 253

pfApplication, 251
calledby npApplication2, 161
calledby npApplication, 160
calledby npEncl, 180
calledby npInterval, 197
calledby npLeftAssoc, 205
calledby npRightAssoc, 204
calledby npSelector, 161
calledby npSynthetic, 196
calledby npTerm, 199
calledby pfBraceBar, 255
calledby pfBrace, 255
calledby pfBracketBar, 255
calledby pfBracket, 255
calledby pfFix, 263
calledby pfFromDom, 265
calledby pfInfApplication, 269
calls pfTree, 251
defun, 251

pfApplication2Sex, 303
calledby pf2Sex1, 298
calls hasOptArgs?, 304

calls opTran, 303
calls pf0TupleParts, 304
calls pf2Sex1, 304
calls pf2Sex, 304
calls pfApplicationArg, 304
calls pfApplicationOp, 303
calls pfOp2Sex, 303
calls pfSuchThat2Sex, 304
calls pfTuple?, 304
uses $insideApplication, 304
uses $insideRule, 304
defun, 303
pfApplication?, 253
calledby macExpand, 220
calledby pf2Sex1, 298
calledby pfCheckItOut, 237
calledby pfCheckMacroOut, 238
calledby pfCollect1?, 240
calledby pfFlattenApp, 239
calledby pfFromDom, 265
calledby pfSexpr,strip, 248
calls pfAbSynOp?, 253
defun, 253
pfApplicationArg, 252
calledby pf0ApplicationArgs, 235
calledby pfApplication2Sex, 304
calledby pfCollectVariable1, 240
calledby pfFlattenApp, 240
calledby pfFromDom, 265
calledby pfSexpr,strip, 248
defun, 252
pfApplicationOp, 252
calledby macApplication, 221
calledby pfApplication2Sex, 303
calledby pfCollect1?, 240
calledby pfFlattenApp, 240
calledby pfFromDom, 265
calledby pfSexpr,strip, 248
defun, 252
pfAssign, 253
calledby npAssignment, 215
calls pfTree, 253
defun, 253
pfAssign?, 254
calledby pf2Sex1, 299
calls pfAbSynOp?, 254
defun, 254
pfAssignLhsItems, 254
calledby pf0AssignLhsItems, 254
defun, 254
pfAssignRhs, 254
calledby pf2Sex1, 299
defun, 254
pfAttribute, 251
calledby npSCategory, 151
calls pfTree, 251
defun, 251
pfBrace, 255
calledby npBraced, 184
calls pfApplication, 255
calls pfIdPos, 255
calls tokPosn, 255
defun, 255
pfBraceBar, 255
calledby npBraced, 184
calls pfApplication, 255
calls pfIdPos, 255
calls tokPosn, 255
defun, 255
pfBracket, 255
calledby npBracked, 184
calls pfApplication, 255
calls pfIdPos, 255
calls tokPosn, 255
defun, 255
pfBracketBar, 255
calledby npBracked, 184
calls pfApplication, 255
calls pfIdPos, 256
calls tokPosn, 256
defun, 255
pfBreak, 256
calledby npBreak, 172
calls pfTree, 256
defun, 256
pfBreak?, 256
calledby pf2Sex1, 299
calls pfAbSynOp?, 256
defun, 256
pfBreakFrom, 256
calledby pf2Sex1, 299
defun, 256

pfCharPosn, 233
  calledby ppos, 355
  calls poCharPosn, 233
  defun, 233
pfCheckArg, 239
  calledby pfCheckMacroOut, 238
  calls pf0TupleParts, 239
  calls pfCheckId, 239
  calls pfListOf, 239
  calls pfTuple?, 239
  defun, 239
pfCheckId, 239
  calledby pfCheckArg, 239
  calledby pfCheckMacroOut, 238
  calls npTrapForm, 239
  calls pfId?, 239
  defun, 239
pfCheckItOut, 237
  calledby npDef, 185
  calls npTrapForm, 237
  calls pf0TupleParts, 237
  calls pfApplication?, 237
  calls pfCollect1?, 237
  calls pfCollectVariable1, 237
  calls pfDefinition?, 237
  calls pfFlattenApp, 237
  calls pfId?, 237
  calls pfListOf, 237
  calls pfNothing, 237
  calls pfTagged?, 237
  calls pfTaggedExpr, 237
  calls pfTaggedTag, 237
  calls pfTaggedToTyped1, 237
  calls pfTaggedToTyped, 237
  calls pfTransformArg, 237
  calls pfTuple?, 237
  calls pfTyped, 237
  defun, 237
pfCheckMacroOut, 238
  calledby npMdef, 162
  calls npTrapForm, 238
  calls pfApplication?, 238
  calls pfCheckArg, 238
  calls pfCheckId, 238
  calls pfFlattenApp, 238
  calls pfId?, 238
  defun, 238
pfCoerceto, 257
  calledby npCoerceTo, 218
  calls pfTree, 257
  defun, 257
pfCoerceto?, 257
  calledby pf2Sex1, 298
  calls pfAbSynOp?, 257
  defun, 257
pfCoercetoExpr, 257
  calledby pf2Sex1, 298
  defun, 257
pfCoercetoType, 257
  calledby pf2Sex1, 298
  defun, 257
pfCollect, 258
  calledby npExpress, 177
  calls pfTree, 258
  defun, 258
pfCollect1?, 240
  calledby pfCheckItOut, 237
  calledby pfFlattenApp, 240
  calledby pfTaggedToTyped1, 242
  calls pfApplication?, 240
  calls pfApplicationOp, 240
  calls pfId?, 240
  calls pfIdSymbol, 240
  defun, 240
pfCollect2Sex, 312
  calledby pf2Sex1, 298
  calls loopIters2Sex, 312
  calls pf2Sex1, 312
  calls pfCollectBody, 312
  calls pfCollectIterators, 312
  calls pfParts, 312
  defun, 312
pfCollect?, 258
  calledby pf2Sex1, 298
  calledby pfCollectArgTran, 315
  calls pfAbSynOp?, 258
  defun, 258
pfCollectArgTran, 315
  calledby pfLambdaTran, 314
  calls pf2sex1, 315
  calls pfCollect?, 315
  calls pfCollectBody, 315

calls pfCollectIterators, 315
calls pfParts, 315
defun, 315
pfCollectBody, 258
calledby pfCollect2Sex, 312
calledby pfCollectArgTran, 315
defun, 258
pfCollectIterators, 258
calledby pfCollect2Sex, 312
calledby pfCollectArgTran, 315
defun, 258
pfCollectVariable1, 240
calledby pfCheckItOut, 237
calledby pfTaggedToTyped1, 242
calls pf0TupleParts, 240
calls pfApplicationArg, 240
calls pfSuch, 240
calls pfTaggedToTyped, 240
calls pfTypedId, 240
calls pfTypedType, 241
calls pfTyped, 240
defun, 240
pfCopyWithPos, 234
calledby macId, 225
calledby pfCopyWithPos, 234
calls pfAbSynOp, 234
calls pfCopyWithPos, 234
calls pfLeaf?, 234
calls pfLeaf, 234
calls pfParts, 234
calls pfTree, 234
calls tokPart, 234
defun, 234
pfDefinition, 259
calledby npDef, 185
calls pfTree, 259
defun, 259
pfDefinition2Sex, 313
calledby pf2Sex1, 299
calls pf0DefinitionLhsItems, 313
calls pf2Sex1, 313
calls pfDefinitionRhs, 313
calls pfLambdaTran, 313
calls systemError, 313
uses $insideApplication, 313
defun, 313
pfDefinition?, 259
calledby pf2Sex1, 299
calledby pfCheckItOut, 237
calledby pfTaggedToTyped1, 242
calls pfAbSynOp?, 259
defun, 259
pfDefinitionLhsItems, 259
calledby pf0DefinitionLhsItems, 260
defun, 259
pfDefinitionRhs, 259
calledby pfDefinition2Sex, 313
defun, 259
pfDo, 260
calledby pfLoop1, 273
calledby pfLp, 274
calls pfTree, 260
defun, 260
pfDo?, 260
calledby pf2Sex1, 299
calls pfAbSynOp?, 260
defun, 260
pfDoBody, 260
calledby pf2Sex1, 299
defun, 260
pfDocument, 244
calledby npParse, 139
calledby npRecoverTrap, 188
calls pfLeaf, 244
defun, 244
pfEnSequence, 261
calledby npEnclosed, 209
calledby npItem, 140
calledby npPDefinition, 181
calledby npPP, 208
calls pfListOf, 261
calls pfSequence, 261
calls pfTuple, 261
defun, 261
pfExit, 261
calledby npPileExit, 214
calls pfTree, 261
defun, 261
pfExit?, 261
calledby pf2Sex1, 298
calls pfAbSynOp?, 261
defun, 261

pfExitCond, 261
calledby pf2Sex1, 298
defun, 261
pfExitExpr, 262
calledby pf2Sex1, 298
defun, 262
pfExport, 262
calledby npExport, 169
calls pfTree, 262
defun, 262
pfExpression, 262
calledby pfSymb, 249
calls ifcar, 262
calls pfLeaf, 262
defun, 262
pfFileName, 234
calledby ppos, 355
calls poFileName, 234
defun, 234
pfFirst, 262
calledby pfLam, 270
calledby pfSourceStok, 241
defun, 262
pfFix, 263
calledby npFix, 164
calls pfApplication, 263
calls pfId, 263
defun, 263
pfFlattenApp, 239
calledby pfCheckItOut, 237
calledby pfCheckMacroOut, 238
calledby pfFlattenApp, 240
calls pfApplication?, 239
calls pfApplicationArg, 240
calls pfApplicationOp, 240
calls pfCollect1?, 240
calls pfFlattenApp, 240
defun, 239
pfForin, 264
calledby npForIn, 176
calls pfTree, 264
defun, 264
pfForin?, 264
calledby pf2Sex1, 298
calls pfAbSynOp?, 264
defun, 264
pfForinLhs, 265
calledby pf0ForinLhs, 265
defun, 265
pfForinWhole, 265
calledby pf2Sex1, 298
defun, 265
pfFree, 263
calledby npFree, 171
calls pfTree, 263
defun, 263
pfFree?, 263
calledby pf2Sex1, 299
calls pfAbSynOp?, 263
defun, 263
pfFreeItems, 264
calledby pf0FreeItems, 264
defun, 264
pfFromDom, 265
calledby npFromdom1, 201
calledby npFromdom, 201
calls pfApplication?, 265
calls pfApplicationArg, 265
calls pfApplicationOp, 265
calls pfApplication, 265
calls pfFromdom, 265
defun, 265
pfFromdom, 266
calledby pfFromDom, 265
calls pfTree, 266
defun, 266
pfFromdom?, 266
calledby pf2Sex1, 298
calls pfAbSynOp?, 266
defun, 266
pfFromdomDomain, 267
calledby pf2Sex1, 298
defun, 267
pfFromdomWhat, 266
calledby pf2Sex1, 298
defun, 266
pfGlobalLinePosn, 233
calledby syIgnoredFromTo, 189
calls poGlobalLinePosn, 233
defun, 233
pfHide, 267
calledby npAngleBared, 184

calls pfTree, 267
defun, 267
pfId, 244
calledby pfFix, 263
calledby pfSuch, 242
calledby pfTaggedToTyped, 287
calls pfLeaf, 244
defun, 244
pfId?, 244
calledby mac0MLambdaApply, 222
calledby mac0SubstituteOuter, 229
calledby macExpand, 220
calledby macMacro, 227
calledby pfCheckId, 239
calledby pfCheckItOut, 237
calledby pfCheckMacroOut, 238
calledby pfCollect1?, 240
calledby pfSexpr,strip, 248
calledby pfTaggedToTyped, 287
calls pfAbSynOp?, 244
defun, 244
pfIdPos, 244
calledby pfBraceBar, 255
calledby pfBrace, 255
calledby pfBracketBar, 256
calledby pfBracket, 255
calls pfLeaf, 244
defun, 244
pfIdSymbol, 245
calledby macId, 225
calledby macLambdaParameterHandling, 229
calledby macMacro, 227
calledby macSubstituteId, 230
calledby pf2Sex1, 298
calledby pfCollect1?, 240
calledby pfInfApplication, 269
calledby pfSexpr,strip, 248
calls tokPart, 245
defun, 245
pfIf, 267
calledby npElse, 194
calledby pfIfThenOnly, 268
calls pfTree, 267
defun, 267
pfIf?, 267
calledby pf2Sex1, 298
calls pfAbSynOp?, 267
defun, 267
pfIfCond, 268
calledby pf2Sex1, 298
calledby pfTweakIf, 288
defun, 268
pfIfElse, 268
calledby pf2Sex1, 298
calledby pfTweakIf, 288
defun, 268
pfIfThen, 268
calledby pf2Sex1, 298
calledby pfTweakIf, 288
defun, 268
pfIfThenOnly, 268
calledby npElse, 194
calls pfIf, 268
calls pfNothing, 268
defun, 268
pfImmediate?
calledby ppos, 355
pfImport, 269
calledby npImport, 178
calls pfTree, 269
defun, 269
pfInfApplication, 269
calledby npInterval, 197
calledby npLeftAssoc, 205
calledby npRightAssoc, 204
calledby npSynthetic, 196
calledby pfSuch, 242
calledby pfTaggedToTyped, 288
calls pfAnd, 269
calls pfApplication, 269
calls pfIdSymbol, 269
calls pfListOf, 269
calls pfOr, 269
calls pfTuple, 269
defun, 269
pfInline, 270
calledby npInline, 172
calls pfTree, 270
defun, 270
pfIterate, 269
calledby npIterate, 172

calls pfTree, 269
defun, 269
pfIterate?, 269
calledby pf2Sex1, 299
calls pfAbSynOp?, 269
defun, 269
pfLam, 270
calledby npLambda, 147
calls pfAbSynOp?, 270
calls pfFirst, 270
calls pfLambda, 270
calls pfNothing, 270
calls pfSecond, 270
defun, 270
pfLambda, 271
calledby pfLam, 270
calledby pfPushBody, 247
calls pfTree, 271
defun, 271
pfLambda2Sex, 315
calledby pf2Sex1, 299
calls pfLambdaTran, 315
defun, 315
pfLambda?, 271
calledby mac0SubstituteOuter, 229
calledby macExpand, 220
calledby macLambdaParameterHandling, 229
calledby pf2Sex1, 299
calledby pfLambdaTran, 314
calls pfAbSynOp?, 271
defun, 271
pfLambdaArgs, 272
calledby pf0LambdaArgs, 272
defun, 272
pfLambdaBody, 271
calledby pfLambdaTran, 314
defun, 271
pfLambdaRets, 271
calledby pfLambdaTran, 314
defun, 271
pfLambdaTran, 314
calledby pfDefinition2Sex, 313
calledby pfLambda2Sex, 315
calls pf0LambdaArgs, 314
calls pf2Sex1, 314
calls pfCollectArgTran, 314
calls pfLambda?, 314
calls pfLambdaBody, 314
calls pfLambdaRets, 314
calls pfNothing?, 314
calls pfTyped?, 314
calls pfTypedId, 314
calls pfTypedType, 314
calls systemError, 314
defun, 314
pfLeaf, 245
calledby macLambdaParameterHandling, 229
calledby pfCopyWithPos, 234
calledby pfDocument, 244
calledby pfExpression, 262
calledby pfIdPos, 244
calledby pfId, 244
calledby pfSymbol, 249
calls ifcar, 245
calls pfNoPosition, 245
calls tokConstruct, 245
defun, 245
pfLeaf?, 245
calledby mac0SubstituteOuter, 229
calledby macLambdaParameterHandling, 229
calledby pfCopyWithPos, 234
calledby pfMapParts, 234
calledby pfSexpr,strip, 248
calledby pfSourcePosition, 236
calledby pfSourceStok, 241
calledby pfSymb, 249
calls pfAbSynOp, 245
defun, 245
pfLeafPosition, 246
calledby macLambdaParameterHandling, 229
calledby pfSourcePosition, 236
calls tokPosn, 246
defun, 246
pfLeafToken, 246
calledby pfLiteral2Sex, 302
calls tokPart, 246
defun, 246
pfLhsRule2Sex, 316

calledby pfRule2Sex, 316
calls pf2Sex1, 316
uses $insideRule, 316
defun, 316
pfLinePosn, 233
calledby ppos, 355
calls poLinePosn, 233
defun, 233
pfListOf, 243
calledby npAssignVariable, 215
calledby npBPileDefinition, 186
calledby npEnclosed, 209
calledby npExpress, 177
calledby npListing, 153
calledby npParse, 139
calledby npRecoverTrap, 188
calledby npSigItemlist, 152
calledby npTypeVariable, 154
calledby npVariable, 211
calledby pfCheckArg, 239
calledby pfCheckItOut, 237
calledby pfEnSequence, 261
calledby pfInfApplication, 269
calledby pfLoop1, 273
calledby pfLp, 274
calledby pfSequenceToList, 237
calledby pfTransformArg, 242
calledby pfTupleListOf, 290
calledby pfTweakIf, 288
calledby pfUnSequence, 291
calls pfTree, 243
defun, 243
pfLiteral2Sex, 302
calledby pf2Sex1, 298
calls float2Sex, 302
calls keyedSystemError, 302
calls pfLeafToken, 302
calls pfLiteralClass, 302
calls pfLiteralString, 302
calls pfSymbolSymbol, 302
uses $insideRule, 302
defun, 302
pfLiteral?, 246
calledby pf2Sex1, 298
calledby pfSexpr,strip, 248
calls pfAbSynOp, 246
defun, 246
pfLiteralClass, 246
calledby pfLiteral2Sex, 302
calls pfAbSynOp, 246
defun, 246
pfLiteralString, 247
calledby pfLiteral2Sex, 302
calledby pfSexpr,strip, 248
calls tokPart, 247
defun, 247
pfLocal, 272
calledby npLocal, 171
calls pfTree, 272
defun, 272
pfLocal?, 272
calledby pf2Sex1, 299
calls pfAbSynOp?, 272
defun, 272
pfLocalItems, 273
calledby pf0LocalItems, 273
defun, 273
pfLoop, 273
calledby pfLoop1, 273
calledby pfLp, 274
calls pfTree, 273
defun, 273
pfLoop1, 273
calledby npLoop, 173
calls pfDo, 273
calls pfListOf, 273
calls pfLoop, 273
defun, 273
pfLoop?, 274
calledby pf2Sex1, 298
calls pfAbSynOp?, 274
defun, 274
pfLoopIterators, 274
defun, 274
pfLp, 274
calledby npLoop, 173
calls pfDo, 274
calls pfListOf, 274
calls pfLoop, 274
defun, 274
pfMacro, 275
calledby macMacro, 227

calledby npMdef, 163
calls pfTree, 275
defun, 275
pfMacro?, 275
calledby macExpand, 220
calledby pf2Sex1, 299
calls pfAbSynOp?, 275
defun, 275
pfMacroLhs, 275
calledby macMacro, 227
defun, 275
pfMacroRhs, 275
calledby macMacro, 227
defun, 275
pfMapParts, 234
calledby macApplication, 221
calledby macExpand, 220
calledby macLambda,mac, 227
calledby macWhere,mac, 226
calls pfAbSynOp, 235
calls pfLeaf?, 234
calls pfParts, 234
calls pfTree, 235
defun, 234
pfMLambda, 276
calledby pfPushMacroBody, 241
calls pfTree, 276
defun, 276
pfMLambda?, 276
calledby macApplication, 221
calledby macLambdaParameterHandling, 229
calledby macMacro, 227
calledby pf2Sex1, 299
calls pfAbSynOp?, 276
defun, 276
pfMLambdaArgs, 276
defun, 276
pfMLambdaBody, 277
calledby mac0GetName, 224
calledby mac0MLambdaApply, 221
defun, 277
pfname, 91
calledby thefname, 91
calls PathnameString, 91
defun, 91
pfNoPosition, 412
calledby pfLeaf, 245
calledby tokPosn, 411
calls poNoPosition, 412
defun, 412
pfNoPosition?, 410
calledby ppos, 355
calledby tokConstruct, 409
calls poNoPosition?, 410
defun, 410
pfNot?, 277
calledby pf2Sex1, 299
calls pfAbSynOp?, 277
defun, 277
pfNotArg, 277
calledby pf2Sex1, 299
defun, 277
pfNothing, 243
calledby macMacro, 227
calledby npAdd, 157
calledby npBreak, 172
calledby npDefaultValue, 193
calledby npIterate, 172
calledby npLocalDecl, 171
calledby npPileBracketed, 186
calledby npPrimary2, 156
calledby npQualType, 179
calledby npReturn, 176
calledby npSignature, 152
calledby npVariableName, 212
calledby npWith, 148
calledby pfAdd, 250
calledby pfCheckItOut, 237
calledby pfIfThenOnly, 268
calledby pfLam, 270
calledby pfPushBody, 247
calledby pfReturnNoName, 283
calledby pfTaggedToTyped1, 242
calledby pfTaggedToTyped, 287
calls pfTree, 243
defun, 243
pfNothing?, 243
calledby macMacro, 227
calledby pf2Sex1, 298
calledby pfLambdaTran, 314
calledby pfTweakIf, 288

calls pfAbSynOp?, 243
defun, 243
pfNovalue, 277
calledby npItem, 140
calledby npVoid, 177
calls pfTree, 277
defun, 277
pfNovalue?, 278
calledby pf2Sex1, 299
calls pfAbSynOp?, 278
defun, 278
pfNovalueExpr, 278
calledby pf2Sex1, 299
defun, 278
pfOp2Sex, 306
calledby pfApplication2Sex, 303
calls pf2Sex1, 306
calls pfSymbol?, 306
calls pmDontQuote?, 306
uses $insideRule, 306
uses $quotedOpList, 306
defun, 306
pfOr, 278
calledby pfInfApplication, 269
calls pfTree, 278
defun, 278
pfOr?, 278
calledby pf2Sex1, 299
calls pfAbSynOp?, 278
defun, 278
pfOrLeft, 279
calledby pf2Sex1, 299
defun, 279
pform
calledby mac0InfiniteExpansion, 223
calledby mac0MLambdaApply, 222
calledby macMacro, 227
calledby phMacro, 219
pfOrRight, 279
calledby pf2Sex1, 299
defun, 279
pfParen, 279
calledby npParened, 183
defun, 279
pfParts, 247
calledby mac0SubstituteOuter, 229
calledby macLambdaParameterHandling, 229
calledby npDefaultDecl, 168
calledby npLocalDecl, 170
calledby npSDefaultItem, 167
calledby npSLocalItem, 170
calledby npSQualTypelist, 179
calledby npSigDecl, 155
calledby npSigItemlist, 152
calledby pf0AssignLhsItems, 254
calledby pf0DefinitionLhsItems, 260
calledby pf0ForinLhs, 265
calledby pf0FreeItems, 264
calledby pf0LambdaArgs, 272
calledby pf0LocalItems, 273
calledby pf0LoopIterators, 274
calledby pf0MLambdaArgs, 276
calledby pf0SequenceArgs, 285
calledby pf0TupleParts, 291
calledby pf0WhereContext, 293
calledby pfCollect2Sex, 312
calledby pfCollectArgTran, 315
calledby pfCopyWithPos, 234
calledby pfMapParts, 234
calledby pfSexpr,strip, 248
calledby pfSourcePosition, 236
calledby pfSourceStok, 241
calledby pfWDec, 292
defun, 247
pfPile, 247
calledby npPileBracketed, 186
defun, 247
pfPretend, 279
calledby npPretend, 217
calls pfTree, 279
defun, 279
pfPretend?, 280
calledby pf2Sex1, 298
calls pfAbSynOp?, 280
defun, 280
pfPretendExpr, 280
calledby pf2Sex1, 298
defun, 280
pfPretendType, 280
calledby pf2Sex1, 298
defun, 280

pfPrintSrcLines
calledby displayParserMacro, 460
pfPushBody, 247
calledby npDef, 185
calledby pfPushBody, 247
calls pfLambda, 247
calls pfNothing, 247
calls pfPushBody, 247
defun, 247
pfPushMacroBody, 241
calledby npMdef, 163
calledby pfPushMacroBody, 241
calls pfMLambda, 241
calls pfPushMacroBody, 241
defun, 241
pfQualType, 280
calledby npQualType, 179
calls pfTree, 280
defun, 280
pfRestrict, 281
calledby npRestrict, 218
calls pfTree, 281
defun, 281
pfRestrict?, 281
calledby pf2Sex1, 299
calls pfAbSynOp?, 281
defun, 281
pfRestrictExpr, 281
calledby pf2Sex1, 299
defun, 281
pfRestrictType, 281
calledby pf2Sex1, 299
defun, 281
pfRetractTo, 282
calledby npColonQuery, 217
calls pfTree, 282
defun, 282
pfReturn, 282
calledby npReturn, 176
calledby pfReturnNoName, 283
calls pfTree, 282
defun, 282
pfReturn?, 282
calledby pf2Sex1, 299
calls pfAbSynOp?, 282
defun, 282
pfReturnExpr, 282
calledby pf2Sex1, 299
defun, 282
pfReturnNoName, 283
calledby npReturn, 176
calls pfNothing, 283
calls pfReturn, 283
defun, 283
pfReturnTyped, 283
calledby npLambda, 147
calls pfTree, 283
defun, 283
pfRhsRule2Sex, 317
calledby pfRule2Sex, 316
calls pf2Sex1, 317
uses $insideRule, 317
defun, 317
pfRule, 283
calledby npSingleRule, 192
calls pfTree, 283
defun, 283
pfRule2Sex, 316
calledby pf2Sex1, 299
calls pfLhsRule2Sex, 316
calls pfRhsRule2Sex, 316
calls pfRuleLhsItems, 316
calls pfRuleRhs, 316
calls ruleLhsTran, 316
calls rulePredicateTran, 316
uses $multiVarPredicateList, 316
uses $predicateList, 316
uses $quotedOpList, 316
defun, 316
pfRule?, 284
calledby pf2Sex1, 299
calls pfAbSynOp?, 284
defun, 284
pfRuleLhsItems, 284
calledby pfRule2Sex, 316
defun, 284
pfRuleRhs, 284
calledby pfRule2Sex, 316
defun, 284
pfSecond, 284
calledby pfLam, 270
defun, 284

pfSequence, 285
calledby npBPileDefinition, 186
calledby pfEnSequence, 261
calls pfTree, 285
defun, 285
pfSequence2Sex, 308
calledby pf2Sex1, 298
calls pf0SequenceArgs, 308
calls pf2Sex1, 308
uses $insideSEQ, 308
defun, 308
pfSequence2Sex0, 308
calledby pfSequence2Sex0, 308
calls pfSequence2Sex0, 308
defun, 308
pfSequence?, 285
calledby pf2Sex1, 298
calledby pfSequenceToList, 236
calledby pfUnSequence, 291
calls pfAbSynOp?, 285
defun, 285
pfSequenceArgs, 285
calledby pf0SequenceArgs, 285
calledby pfSequenceToList, 236
defun, 285
pfSequenceToList, 236
calledby npDefinition, 165
calls pfListOf, 237
calls pfSequence?, 236
calls pfSequenceArgs, 236
defun, 236
pfSexpr, 248
calledby pfSymb, 249
calls pfSexpr,strip, 248
defun, 248
pfSexpr,strip, 248
calledby pfSexpr,strip, 248
calledby pfSexpr, 248
calls pf0TupleParts, 248
calls pfAbSynOp, 248
calls pfApplication?, 248
calls pfApplicationArg, 248
calls pfApplicationOp, 248
calls pfId?, 248
calls pfIdSymbol, 248
calls pfLeaf?, 248
calls pfLiteral?, 248
calls pfLiteralString, 248
calls pfParts, 248
calls pfSexpr,strip, 248
calls pfTuple?, 248
calls tokPart, 248
defun, 248
pfSourcePosition, 236
calledby mac0ExpandBody, 222
calledby mac0MLambdaApply, 221
calledby macId, 225
calledby macMacro, 227
calledby pfSourcePosition, 236
calls pfLeaf?, 236
calls pfLeafPosition, 236
calls pfParts, 236
calls pfSourcePosition, 236
calls poNoPosition?, 236
uses $nopos, 236
defun, 236
pfSourceStok, 241
calledby npTrapForm, 210
calledby pfSourceStok, 241
calls pfFirst, 241
calls pfLeaf?, 241
calls pfParts, 241
calls pfSourceStok, 241
defun, 241
pfSpread, 237
calledby npDefaultDecl, 168
calledby npLocalDecl, 170
calledby npSigDecl, 155
calls pfTyped, 237
defun, 237
pfSuch, 242
calledby pfCollectVariable1, 240
calledby pfTaggedToTyped, 287
calls pfId, 242
calls pfInfApplication, 242
defun, 242
pfSuchthat, 286
calledby npSuchThat, 174
calls pfTree, 286
defun, 286
pfSuchThat2Sex, 305
calledby pfApplication2Sex, 304

calls pf0TupleParts, 305
calls pf2Sex1, 305
calls pf2Sex, 305
uses $predicateList, 305
defun, 305
pfSuchthat?, 286
calledby pf2Sex1, 299
calls pfAbSynOp?, 286
defun, 286
pfSuchthatCond, 286
calledby pf2Sex1, 299
defun, 286
pfSymb, 249
calledby npConstTok, 182
calledby npDDInfKey, 206
calledby npInfixOperator, 158
calls ifcar, 249
calls pfExpression, 249
calls pfLeaf?, 249
calls pfSexpr, 249
calls pfSymbol, 249
calls tokPart, 249
defun, 249
pfSymbol, 249
calledby pfSymb, 249
calls ifcar, 249
calls pfLeaf, 249
defun, 249
pfSymbol?, 250
calledby pf2Sex1, 298
calledby pfOp2Sex, 306
calls pfAbSynOp?, 250
defun, 250
pfSymbolSymbol, 250
calledby pf2Sex1, 298
calledby pfLiteral2Sex, 302
calls tokPart, 250
defun, 250
pfTagged, 286
calledby npTagged, 216
calls pfTree, 286
defun, 286
pfTagged?, 287
calledby pf2Sex1, 298
calledby pfCheckItOut, 237
calledby pfTaggedToTyped, 287
calls pfAbSynOp?, 287
defun, 287
pfTaggedExpr, 287
calledby pf2Sex1, 298
calledby pfCheckItOut, 237
calledby pfTaggedToTyped, 287
defun, 287
pfTaggedTag, 287
calledby pf2Sex1, 298
calledby pfCheckItOut, 237
calledby pfTaggedToTyped, 287
defun, 287
pfTaggedToTyped, 287
calledby pfCheckItOut, 237
calledby pfCollectVariable1, 240
calledby pfTaggedToTyped1, 242
calls pfId?, 287
calls pfId, 287
calls pfInfApplication, 288
calls pfNothing, 287
calls pfSuch, 287
calls pfTagged?, 287
calls pfTaggedExpr, 287
calls pfTaggedTag, 287
calls pfTyped, 287
defun, 287
pfTaggedToTyped1, 242
calledby pfCheckItOut, 237
calledby pfTransformArg, 242
calls pfCollect1?, 242
calls pfCollectVariable1, 242
calls pfDefinition?, 242
calls pfNothing, 242
calls pfTaggedToTyped, 242
calls pfTyped, 242
defun, 242
pfTransformArg, 242
calledby pfCheckItOut, 237
calls pf0TupleParts, 242
calls pfListOf, 242
calls pfTaggedToTyped1, 242
calls pfTuple?, 242
defun, 242
pfTree, 250
calledby pfAdd, 250
calledby pfAnd, 251

calledby pfApplication, 251
calledby pfAssign, 253
calledby pfAttribute, 251
calledby pfBreak, 256
calledby pfCoerceto, 257
calledby pfCollect, 258
calledby pfCopyWithPos, 234
calledby pfDefinition, 259
calledby pfDo, 260
calledby pfExit, 261
calledby pfExport, 262
calledby pfForin, 264
calledby pfFree, 263
calledby pfFromdom, 266
calledby pfHide, 267
calledby pfIf, 267
calledby pfImport, 269
calledby pfInline, 270
calledby pfIterate, 269
calledby pfLambda, 271
calledby pfListOf, 243
calledby pfLocal, 272
calledby pfLoop, 273
calledby pfMLambda, 276
calledby pfMacro, 275
calledby pfMapParts, 235
calledby pfNothing, 243
calledby pfNovalue, 277
calledby pfOr, 278
calledby pfPretend, 279
calledby pfQualType, 280
calledby pfRestrict, 281
calledby pfRetractTo, 282
calledby pfReturnTyped, 283
calledby pfReturn, 282
calledby pfRule, 283
calledby pfSequence, 285
calledby pfSuchthat, 286
calledby pfTagged, 286
calledby pfTuple, 290
calledby pfTweakIf, 288
calledby pfTyped, 288
calledby pfTyping, 289
calledby pfWDeclare, 292
calledby pfWhere, 292
calledby pfWhile, 293
calledby pfWith, 294
calledby pfWrong, 294
defun, 250
pfTuple, 290
calledby npEnclosed, 209
calledby pfEnSequence, 261
calledby pfInfApplication, 269
calledby pfTupleListOf, 290
calls pfTree, 290
defun, 290
pfTuple?, 290
calledby pf0FlattenSyntacticTuple, 235
calledby pf2Sex1, 298
calledby pfApplication2Sex, 304
calledby pfCheckArg, 239
calledby pfCheckItOut, 237
calledby pfSexpr,strip, 248
calledby pfTransformArg, 242
calls pfAbSynOp?, 290
defun, 290
pfTupleListOf, 290
calledby npTuple, 144
calls pfListOf, 290
calls pfTuple, 290
defun, 290
pfTupleParts, 291
calledby pf0TupleParts, 291
defun, 291
pfTweakIf, 288
calledby npWConditional, 193
calls pfIfCond, 288
calls pfIfElse, 288
calls pfIfThen, 288
calls pfListOf, 288
calls pfNothing?, 288
calls pfTree, 288
defun, 288
pfTyped, 288
calledby npDecl, 212
calledby npVariableName, 212
calledby pfCheckItOut, 237
calledby pfCollectVariable1, 240
calledby pfSpread, 237
calledby pfTaggedToTyped1, 242
calledby pfTaggedToTyped, 287
calls pfTree, 288

defun, 288
pfTyped?, 289
calledby pf2Sex1, 299
calledby pfLambdaTran, 314
calls pfAbSynOp?, 289
defun, 289
pfTypedId, 289
calledby macLambdaParameterHandling, 229
calledby pf2Sex1, 299
calledby pfCollectVariable1, 240
calledby pfLambdaTran, 314
defun, 289
pfTypedType, 289
calledby pf2Sex1, 299
calledby pfCollectVariable1, 241
calledby pfLambdaTran, 314
defun, 289
pfTyping, 289
calledby npTyping, 166
calls pfTree, 289
defun, 289
pfUnSequence, 291
calledby npCategoryL, 150
calledby npDefaultItemlist, 167
calledby npLocalItemlist, 169
calledby npQualTypelist, 178
calls pf0SequenceArgs, 291
calls pfAppend, 291
calls pfListOf, 291
calls pfSequence?, 291
defun, 291
pfWDec, 291
calledby npSignature, 152
calls pfParts, 292
calls pfWDeclare, 291
defun, 291
pfWDeclare, 292
calledby pfWDec, 291
calls pfTree, 292
defun, 292
pfWhere, 292
calledby npLetQualified, 165
calledby npQualified, 145
calls pfTree, 292
defun, 292
pfWhere?, 292
calledby macExpand, 220
calledby pf2Sex1, 299
calls pfAbSynOp?, 292
defun, 292
pfWhereContext, 293
calledby pf0WhereContext, 293
defun, 293
pfWhereExpr, 293
calledby pf2Sex1, 299
defun, 293
pfWhile, 293
calledby npWhile, 175
calls pfTree, 293
defun, 293
pfWhile?, 294
calledby pf2Sex1, 298
calls pfAbSynOp?, 294
defun, 294
pfWhileCond, 294
calledby pf2Sex1, 298
defun, 294
pfWith, 294
calledby npWith, 148
calls pfTree, 294
defun, 294
pfWrong, 294
calledby npParse, 139
calledby npRecoverTrap, 188
calls pfTree, 294
defun, 294
pfWrong?, 295
calledby pf2Sex1, 299
calls pfAbSynOp?, 295
defun, 295
phInterpret, 67
calls intInterpretPform, 67
calls ncEltQ, 67
calls ncPutQ, 67
defun, 67
phIntReportMsgs, 66
calls ncEltQ, 66
calls ncPutQ, 66
calls processMsgList, 66
uses $erMsgToss, 66
defun, 66

phMacro, 219
calls macroExpanded, 219
calls ncEltQ, 219
calls ncPutQ, 219
calls pform, 219
defun, 219
phParse, 66
calls ncPutQ, 66
defun, 66
pileCforest, 338
calledby insertpile, 333
calledby pileCtree, 338
calls enPile, 338
calls separatePiles, 338
calls tokPart, 338
defun, 338
pileColumn, 335
calledby eqpileTree, 337
calledby pileTree, 335
calls tokPosn, 335
defun, 335
pileCtree, 338
calledby pileForests, 336
calls dqAppend, 338
calls pileCforest, 338
defun, 338
pileForest, 336
calledby pileForests, 335
calls pileForest1, 336
calls pileTree, 336
defun, 336
pileForest1, 336
calledby pileForest1, 337
calledby pileForest, 336
calls eqpileTree, 336
calls pileForest1, 337
defun, 336
pileForests, 335
calledby eqpileTree, 337
calledby pileForests, 336
calledby pileTree, 335
calls npNull, 336
calls pileCtree, 336
calls pileForests, 336
calls pileForest, 335
defun, 335
pilePlusComment, 334
calledby insertpile, 333
calledby pilePlusComments, 334
calls tokType, 334
defun, 334
pilePlusComments, 334
calledby insertpile, 333
calledby pilePlusComments, 334
calls npNull, 334
calls pilePlusComments, 334
calls pilePlusComment, 334
defun, 334
pileTree, 335
calledby insertpile, 333
calledby pileForest, 336
calls npNull, 335
calls pileColumn, 335
calls pileForests, 335
defun, 335
placep
calledby unwritable?, 602
calledby writify,writifyInner, 603
pmDontQuote?, 307
calledby pfOp2Sex, 306
defun, 307
pname, 1039
calledby clearCmdParts, 501
calledby coerceTraceArgs2E, 854
calledby coerceTraceFunValue2E, 857
calledby displayType, 456
calledby displayValue, 455
calledby fixObjectForPrinting, 452
calledby gensymInt, 612
calledby getAliasIfTracedMapParameter, 880
calledby getStFromMsg, 352
calledby hasOption, 447
calledby isSharpVarWithNum, 876
calledby letPrint2, 877
calledby letPrint3, 878
calledby letPrint, 875
calledby mac0InfiniteExpansion,name, 224
calledby newHelpSpad2Cmd, 569
calledby npMissing, 149
calledby reportUndo, 918
calledby rwrite, 600, 601

calledby selectOption, 475
calledby setFortDir, 717
calledby setFortTmpDir, 715
calledby setOutputCharacters, 758
calledby stupidIsSpadFunction, 896
calledby undo, 912
defun, 1039
poCharPosn, 375
calledby compareposns, 368
calledby pfCharPosn, 233
calledby posPointers, 376
calledby processChPosesForOneLine, 374
calledby putFTText, 377
defun, 375
poFileName, 358
calledby decideHowMuch, 357
calledby pfFileName, 234
calls lnFileName, 358
calls poGetLineObject, 358
defun, 358
poGetLineObject, 359
calledby poFileName, 358
calledby poGlobalLinePosn, 72
calledby poLinePosn, 359
calledby poPosImmediate?, 358
defun, 359
poGlobalLinePosn, 72
calledby compareposns, 368
calledby listDecideHowMuch, 359
calledby makeMsgFromLine, 369
calledby ncloopDQlines, 71
calledby pfGlobalLinePosn, 233
calledby processMsgList, 367
calledby thisPosIsEqual, 372
calledby thisPosIsLess, 372
calls lnGlobalNum, 72
calls ncBug, 72
calls poGetLineObject, 72
defun, 72
poLinePosn, 359
calledby decideHowMuch, 357
calledby makeMsgFromLine, 369
calledby pfLinePosn, 233
calls lnLocalNum, 359
calls poGetLineObject, 359
defun, 359
poNopos?, 358
calledby decideHowMuch, 357
calledby erMsgSep, 368
calledby listDecideHowMuch, 359
calledby poPosImmediate?, 358
calledby thisPosIsEqual, 372
calledby thisPosIsLess, 372
defun, 358
poNoPosition, 412
calledby pfNoPosition, 412
uses $nopos, 412
defun, 412
poNoPosition?, 411
calledby pfNoPosition?, 410
calledby pfSourcePosition, 236
calls eqcar, 411
defun, 411
poPosImmediate?, 358
calledby decideHowMuch, 357
calledby listDecideHowMuch, 359
calls lnImmediate?, 358
calls poGetLineObject, 358
calls poNopos?, 358
defun, 358
porigin, 88
calledby inclmsgFileCycle, 92
calledby ppos, 355
calls stringp, 88
defun, 88
posend, 126
calledby scanW, 125
defun, 126
posPointers, 376
calledby processChPosesForOneLine, 374
calls IFCAR, 376
calls getMsgPos2, 376
calls getMsgPos, 376
calls insertPos, 376
calls poCharPosn, 376
uses getMsgFTTag, 376
defun, 376
poundsign
calledby dewritify,dewritifyInner, 608
calledby displaySetVariableSettings, 649
calledby getTraceOptions, 842
calledby isDomainOrPackage, 865

calledby newHelpSpad2Cmd, 569
calledby set1, 800
calledby setOutputLibrary, 656
calledby trace1, 838
calledby traceReply, 889
pp
calledby reportUndo, 918
pp2Cols
calledby filterAndFormatConstructors, 934
ppos, 355
calledby getPosStL, 354
calls pfCharPosn, 355
calls pfFileName, 355
calls pfImmediate?, 355
calls pfLinePosn, 355
calls pfNoPosition?, 355
calls porigin, 355
defun, 355
pquit, 630
calls pquitSpad2Cmd, 630
defun, 630
pquit help page, 629
manpage, 629
pquitSpad2Cmd, 630
calledby pquit, 630
calls quitSpad2Cmd, 630
uses $quitCommandType, 630
defun, 630
pred2English
calledby displayCondition, 461
prefix2String
calledby displayMode, 463
calledby displayProperties, 458
calledby displayType, 456
calledby displayValue, 455
calledby evalDomain, 903
calledby spadUntrace, 886
calledby traceReply, 889
previousInterpreterFrame, 557
calledby frameSpad2Cmd, 562
calls updateCurrentInterpreterFrame, 557
calls updateFromCurrentInterpreterFrame, 557
uses $interpreterFrameRing, 557
defun, 557
print
calledby letPrint2, 877
calledby letPrint3, 878
printAsTeX, 61
calledby printTypeAndTimeSaturn, 60
uses $texOutputStream, 61
defun, 61
printDashedLine
calledby spadTrace, 867
printLabelledList, 470
calledby printSynonyms, 470
calls blankList, 470
calls concat, 470
calls entryWidth, 470
calls fillerSpaces, 470
calls sayBrightly, 470
calls sayMessage, 470
calls substring, 470
defun, 470
printStatisticsSummary, 57
calledby recordAndPrint, 56
calls sayKeyedMsg, 57
calls statisticsSummary, 57
uses $collectOutput, 57
defun, 57
printStorage, 58
calledby recordAndPrint, 56
calls makeLongSpaceString, 58
uses $collectOutput, 58
uses $interpreterTimedClasses, 58
uses $interpreterTimedNames, 58
defun, 58
printSynonyms, 470
calledby npProcessSynonym, 469
calledby synonymSpad2Cmd, 824
calledby whatSpad2Cmd, 930
calls centerAndHighlight, 470
calls filterListOfStringsWithFn, 470
calls printLabelledList, 470
calls specialChar, 470
calls synonymsForUserLevel, 470
uses $CommandSynonymAlist, 470
uses $linelength, 470
defun, 470
printTypeAndTime, 58
calledby recordAndPrint, 56
calls printTypeAndTimeNormal, 58

calls printTypeAndTimeSaturn, 58
uses $saturn, 58
defun, 58
printTypeAndTimeNormal, 59
calledby printTypeAndTime, 58
calls justifyMyType, 59
calls makeLongTimeString, 59
calls msgText, 59
calls objMode, 59
calls objNewWrap, 59
calls qcar, 59
calls retract, 59
calls sameUnionBranch, 59
calls sayKeyedMsg, 59
uses $collectOutput, 59
uses $interpreterTimedClasses, 59
uses $interpreterTimedNames, 59
uses $outputLines, 59
uses $printTimeIfTrue, 59
uses $printTypeIfTrue, 59
defun, 59
printTypeAndTimeSaturn, 60
calledby printTypeAndTime, 58
calls devaluate, 60
calls form2StringAsTeX, 60
calls makeLongTimeString, 60
calls printAsTeX, 60
uses $interpreterTimedClasses, 60
uses $interpreterTimedNames, 60
uses $printTimeIfTrue, 60
uses $printTypeIfTrue, 60
defun, 60
prior-token
usedby token-stack-show, 961
probeName, 974
defun, 974
processChPosesForOneLine, 374
calledby queueUpErrors, 370
calls getMsgFTTag?, 374
calls getMsgPos, 374
calls getMsgPrefix, 374
calls makeLeaderMsg, 374
calls poCharPosn, 374
calls posPointers, 374
calls putFTText, 374
calls setMsgPrefix, 374
calls size, 374
calls strconc, 374
uses $preLength, 374
defun, 374
processInteractive, 49
calledby intInterpretPform, 67
calls clrhash, 50
calls initializeTimedNames, 49
calls processInteractive1, 50
calls qcar, 50
calls reportInstantiations, 50
calls updateHist, 50
calls writeHistModesAndValues, 50
uses $Coerce, 50
uses $ProcessInteractiveValue, 50
uses $StreamFrame, 50
uses $analyzingMapList, 50
uses $compErrorMessageStack, 50
uses $compilingLoop, 50
uses $compilingMap, 50
uses $declaredMode, 50
uses $defaultFortVar, 50
uses $domPvar, 50
uses $fortVar, 50
uses $freeVars, 50
uses $inRetract, 50
uses $instantCanCoerceCount, 50
uses $instantCoerceCount, 50
uses $instantMmCondCount, 50
uses $instantRecord, 50
uses $interpOnly, 50
uses $interpreterTimedClasses, 50
uses $interpreterTimedNames, 50
uses $lastLineInSEQ, 50
uses $localVars, 50
uses $mapList, 50
uses $minivectorCode, 50
uses $minivectorNames, 50
uses $minivector, 50
uses $op, 50
uses $reportInstantiations, 50
uses $timeGlobalName, 50
uses $whereCacheList, 50
defun, 49
processInteractive1, 52
calledby processInteractive, 50

calls interpretTopLevel, 52
calls objMode, 52
calls objValUnwrap, 52
calls recordAndPrint, 52
calls recordFrame, 52
calls startTimingProcess, 52
calls stopTimingProcess, 52
uses $InteractiveFrame, 52
uses $ProcessInteractiveValue, 52
uses $e, 52
defun, 52
processKeyedError, 351
calledby ncBug, 366
calledby ncHardError, 350
calledby ncSoftError, 349
calls CallerName, 351
calls getMsgKey, 351
calls getMsgPrefix?, 351
calls getMsgTag?, 351
calls msgImPr?, 351
calls msgOutputter, 351
calls sayBrightly, 351
uses $ncMsgList, 351
defun, 351
processMsgList, 367
calledby phIntReportMsgs, 66
calls erMsgSort, 367
calls getMsgPos, 367
calls listOutputter, 367
calls makeMsgFromLine, 367
calls poGlobalLinePosn, 367
calls queueUpErrors, 367
uses $noRepList, 367
uses $outputList, 367
defun, 367
processSynonymLine, 827
calledby npProcessSynonym, 469
calledby synonymSpad2Cmd, 824
calls processSynonymLine,removeKeyFromLine, 827
defun, 827
processSynonymLine,removeKeyFromLine, 826
calledby processSynonymLine, 827
calls dropLeadingBlanks, 826
calls maxindex, 826
defun, 826
processSynonyms, 33
calledby doSystemCommand, 442
calledby intProcessSynonyms, 33
calledby processSynonyms, 33
calls concat, 33
calls lassoc, 33
calls processSynonyms, 33
calls rplacstr, 33
calls size, 33
calls strconc, 33
calls string2id-n, 33
calls strpos, 33
calls substring, 33
uses $CommandSynonymAlist, 34
uses line, 34
defun, 33
protectedEVAL, 47
calledby serverReadLine, 44
calls resetStackLimits, 47
calls sendHTErrorSignal, 47
defun, 47
prTraceNames, 888
calls exit, 888
calls prTraceNames,fn, 888
calls seq, 888
uses /tracenames, 888
defun, 888
prTraceNames,fn, 888
calledby prTraceNames, 888
calls devaluate, 888
calls exit, 888
calls isDomainOrPackage, 888
calls qcar, 888
calls qcdr, 888
calls seq, 888
defun, 888
pspacers, 849
calls bright, 849
calls concat, 849
calls sayBrightly, 849
uses /spacelist, 849
defun, 849
ptimers, 849
calledby trace1, 838
calls bright, 849
calls concat, 849

calls float, 849
calls quotient, 849
calls sayBrightly, 849
uses /timerlist, 849
defun, 849
punctuation?, 115
calledby scanToken, 111
defun, 115
put, 1033
defun, 1033
putalist
calledby interpFunctionDepAlists, 461
calledby npProcessSynonym, 469
calledby synonymSpad2Cmd, 824
putDatabaseStuff, 363
calledby msgCreate, 346
calls getMsgInfoFromKey, 363
calls setMsgText, 363
calls setMsgUnforcedAttrList, 363
defun, 363
putFTText, 377
calledby processChPosesForOneLine, 374
calls getMsgFTTag?, 377
calls getMsgPos2, 377
calls getMsgPos, 377
calls getMsgText, 377
calls poCharPosn, 377
calls setMsgText, 377
defun, 377
putHist, 586
calledby importFromFrame, 559
calledby recordAndPrint, 56
calledby restoreHistory, 593
calledby undoChanges, 589
calledby undoFromFile, 590
calledby undoInCore, 589
calledby writeHistModesAndValues, 599
calls get, 586
calls putIntSymTab, 587
calls recordNewValue, 587
calls recordOldValue, 586
uses $HiFiAccess, 587
defun, 586
putIntSymTab
calledby putHist, 587
putTarget
calledby evaluateType1, 908
calledby interpret1, 54
pvarPredTran, 320
calledby rulePredicateTran, 317
defun, 320

qassq
calledby getMsgCatAttr, 356
calledby ncEltQ, 414
calledby ncPutQ, 415
calledby setMsgCatlessAttr, 363
calledby setMsgUnforcedAttr, 365
calledby tokPosn, 411
qcar
calledby /tracereply, 884
calledby ?t, 893
calledby NRTevalDomain, 1064
calledby ScanOrPairVec,ScanOrInner, 611
calledby abbreviationsSpad2Cmd, 479
calledby dewritify,dewritifyInner, 608
calledby displayProperties, 457
calledby evaluateType1, 908
calledby evaluateType, 906
calledby frameSpad2Cmd, 562
calledby funfind,LAM, 864
calledby getTraceOption, 844
calledby hasPair, 881
calledby mkEvalable, 903
calledby ncAlist, 413
calledby ncTag, 413
calledby prTraceNames,fn, 888
calledby printTypeAndTimeNormal, 59
calledby processInteractive, 50
calledby reportOperations, 807
calledby reportOpsFromUnitDirectly, 814
calledby reportSpadTrace, 882
calledby restoreHistory, 593
calledby retract, 1049
calledby selectOption, 476
calledby setExposeAddConstr, 692
calledby setExposeAddGroup, 690
calledby setExposeAdd, 690
calledby setExposeDropConstr, 695
calledby setExposeDropGroup, 694
calledby setExposeDrop, 693
calledby setExpose, 689

calledby setInputLibrary, 659
calledby setOutputAlgebra, 754
calledby setOutputCharacters, 758
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby showSpad2Cmd, 806
calledby spadClosure?, 607
calledby spadReply,printName, 884
calledby trace1, 838
calledby traceReply, 889
calledby traceSpad2Cmd, 837
calledby transOnlyOption, 850
calledby undoSteps, 920
calledby undo, 912
calledby untraceDomainConstructor,keepTraced, 872
calledby whatSpad2Cmd,fixpat, 929
calledby writify,writifyInner, 603

qcdr
calledby ?t, 893
calledby ScanOrPairVec,ScanOrInner, 611
calledby abbreviationsSpad2Cmd, 479
calledby dewritify,dewritifyInner, 608
calledby displayProperties, 457
calledby evaluateType1, 908
calledby evaluateType, 906
calledby frameSpad2Cmd, 562
calledby getTraceOption, 844
calledby hasPair, 881
calledby mkEvalable, 903
calledby ncAlist, 414
calledby prTraceNames,fn, 888
calledby readHiFi, 597
calledby restoreHistory, 593
calledby selectOption, 475
calledby setExposeAdd, 690
calledby setExposeDrop, 693
calledby setExpose, 689
calledby setInputLibrary, 659
calledby setOutputAlgebra, 754
calledby setOutputCharacters, 758
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby spadClosure?, 607
calledby trace1, 838
calledby traceSpad2Cmd, 837
calledby transOnlyOption, 850
calledby undoSteps, 920
calledby undo, 912
calledby writify,writifyInner, 603

qcsize
calledby isSharpVarWithNum, 876

qenum, 1040
calledby scanEsc, 121
calledby scanExponent, 124
calledby scanIgnoreLine, 110
calledby scanInsert, 136
calledby scanNumber, 129
calledby scanToken, 111
calledby scanW, 125
calledby spleI1, 120
calledby startsComment?, 113
calledby startsNegComment?, 114
calledby substringMatch, 117
defun, 1040

qrplaca
calledby dewritify,dewritifyInner, 608
calledby writify,writifyInner, 603

qrplacd
calledby dewritify,dewritifyInner, 608
calledby writify,writifyInner, 603

qsabsval, 1054
defmacro, 1054

qsadd1, 1053
defmacro, 1053

qsdifference, 1052
defmacro, 1052

qsetvelt
calledby dewritify,dewritifyInner, 608
calledby writify,writifyInner, 603

qslessp, 1053
calledby trace1, 838
defmacro, 1053

qsmax, 1055

defmacro, 1055
qsmin, 1055
defmacro, 1055
qsminus, 1053
defmacro, 1053
qsoddp, 1054
defmacro, 1054
qsplus, 1054
defmacro, 1054
qsquotient, 1052
defun, 1052
qsremainder, 1052
defun, 1052
qssub1, 1053
defmacro, 1053
qstimes, 1054
defmacro, 1054
qszerop, 1055
defmacro, 1055
queryClients, 506
calledby close, 507
calls sockGetInt, 506
calls sockSendInt, 506
uses $QueryClients, 506
uses $SessionManager, 506
defun, 506
queryUserKeyedMsg
calledby close, 506
calledby historySpad2Cmd, 578
calledby importFromFrame, 559
calledby listConstructorAbbreviations, 480
calledby quitSpad2Cmd, 634
calledby yesanswer, 533
queueUpErrors, 370
calledby processMsgList, 367
calls processChPosesForOneLine, 370
uses $outputList, 370
defun, 370
quit, 634
calls quitSpad2Cmd, 634
defun, 634
quit help page, 633
manpage, 633
quitSpad2Cmd, 634
calledby pquitSpad2Cmd, 630
calledby quit, 634
calls leaveScratchpad, 634
calls queryUserKeyedMsg, 634
calls sayKeyedMsg, 634
calls string2id-n, 634
calls tersyscommand, 634
calls upcase, 634
uses $quitCommandType, 634
defun, 634
QUOTIENT
calledby Else?, 80
calledby Elseif?, 79
quotient
calledby If?, 79
calledby Top?, 79
calledby ptimers, 849
quotient2, 1073
defun, 1073
qv32len, 1048
defmacro, 1048
qvelt
calledby dewritify,dewritifyInner, 608
calledby writify,writifyInner, 603
qvmaxindex
calledby dewritify,dewritifyInner, 608
calledby writify,writifyInner, 603

random, 1073
defun, 1073
rassoc
calledby rassocSub, 861
calledby saveMapSig, 843
rassocSub, 861
calledby ?t, 893
calledby isSubForRedundantMapName, 863
calledby traceReply, 889
calls rassoc, 861
defun, 861
rdefinstream, 1059
calls rdefiostream, 1059
defun, 1059
rdefiostream
calledby rdefinstream, 1059
calledby rdefoutstream, 1060
calledby readHiFi, 597
calledby saveHistory, 591
calledby setHistoryCore, 581

calledby writeHiFi, 598
rdefoutstream, 1060
calls rdefiostream, 1060
defun, 1060
rdigit?, 131
calls strpos, 131
defun, 131
read, 638
calledby undo, 912
calls readSpad2Cmd, 638
defun, 638
read help page, 637
manpage, 637
read-line
calledby serverReadLine, 44
readHiFi, 597
calledby fetchOutput, 596
calledby restoreHistory, 593
calledby setHistoryCore, 581
calledby showInOut, 596
calledby showInput, 595
calledby undoFromFile, 590
calledby undoInCore, 589
calledby writeInputLines, 583
calls assoc, 597
calls histFileName, 597
calls keyedSystemError, 597
calls object2Identifier, 597
calls qcdr, 597
calls rdefiostream, 597
calls rshut, 597
calls spadrread, 597
uses $internalHistoryTable, 597
uses $useInternalHistoryTable, 597
defun, 597
readSpad2Cmd, 638
calledby read, 638
calls /read, 638
calls findfile, 638
calls makePathname, 638
calls member, 638
calls mergePathnames, 638
calls namestring, 638
calls optionError, 638
calls pathnameName, 638
calls pathnameTypeId, 638
calls pathname, 638
calls selectOptionLC, 638
calls throwKeyedMsg, 638
calls upcase, 638
uses /editfile, 639
uses $InteractiveMode, 638
uses $UserLevel, 638
uses $findfile, 638
uses $options, 638
defun, 638
readSpadProfileIfThere, 951
calledby restart, 17
uses /editfile, 951
defun, 951
reclaim, 39
calledby clearCmdCompletely, 498
defun, 39
recordAndPrint, 56
calledby processInteractive1, 52
calls mkCompanionPage, 56
calls objNewWrap, 56
calls output, 56
calls printStatisticsSummary, 56
calls printStorage, 56
calls printTypeAndTime, 56
calls putHist, 56
calls recordAndPrintTest, 56
uses $EmptyMode, 56
uses $HTCompanionWindowID, 56
uses $QuietCommand, 56
uses $Void, 56
uses $algebraOutputStream, 56
uses $collectOutput, 56
uses $e, 56
uses $mkTestFlag, 56
uses $mkTestOutputType, 56
uses $outputMode, 56, 57
uses $printAnyIfTrue, 57
uses $printStatisticsSummaryIfTrue, 56
uses $printStorageIfTrue, 56
uses $printTimeIfTrue, 56
uses $printTypeIfTrue, 56
uses $printVoidIfTrue, 56
uses $runTestFlag, 56
defun, 56
recordAndPrintTest

calledby recordAndPrint, 56
recordFrame, 913
calledby processInteractive1, 52
calledby undoSteps, 920
calls diffAlist, 913
calls exit, 913
calls kar, 913
calls seq, 913
uses $InteractiveFrame, 913
uses $frameRecord, 913
uses $previousBindings, 913
uses $undoFlag, 913
defun, 913
recordNewValue, 587
calledby clearCmdParts, 501
calledby putHist, 587
calledby undoFromFile, 590
calls recordNewValue0, 587
calls startTimingProcess, 587
calls stopTimingProcess, 587
defun, 587
recordNewValue0, 587
calledby recordNewValue, 587
calls assq, 587
uses $HistRecord, 587
defun, 587
recordOldValue, 588
calledby clearCmdParts, 501
calledby putHist, 586
calledby undoFromFile, 590
calls recordOldValue0, 588
calls startTimingProcess, 588
calls stopTimingProcess, 588
defun, 588
recordOldValue0, 588
calledby recordOldValue, 588
calls assq, 588
uses $HistList, 588
defun, 588
reduce-stack-clear
calledby ioclear, 962
redundant, 372
calls msgNoRep?, 372
calls sameMsg?, 372
uses $noRepList, 372
defun, 372
refvecp
calledby isDomainOrPackage, 865
calledby spadTrace, 866
remainder
calledby KeepPart?, 80
calledby SkipEnd?, 80
calledby SkipPart?, 81
remainder2, 1072
defun, 1072
remalist
calledby clearParserMacro, 449
remdup
calledby clearCmdParts, 501
calledby displayMacros, 534
calledby displayOperationsFromLisplib, 812
calledby displayProperties, 457
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
remFile, 355
calledby getPosStL, 354
calls IFCDR, 355
defun, 355
remLine, 360
calledby getPosStL, 354
calls IFCAR, 355
defun, 360
removeAttributes
calledby getMsgInfoFromKey, 364
removeOption, 851
calledby spadTrace, 866
defun, 851
remover, 887
calledby remover, 887
calledby spadUntrace, 886
calls remover, 887
defun, 887
removeTracedMapSigs, 854
calledby untrace, 852
uses $tracedMapSignatures, 854
defun, 854
removeUndoLines, 923
calls charPosition, 923
calls concat, 924
calls exit, 923
calls maxindex, 924

calls seq, 923
calls spaddifference, 924
calls stringPrefix?, 923
calls substring, 923
calls trimString, 923
calls undoCount, 924
uses $IOindex, 924
uses $currentLine, 924
defun, 923
removeZeroOneDestructively
calledby reportOperations, 808
remprop
calledby loadLib, 1030
rempropI
calledby restoreHistory, 593
rep, 369
calledby makeMsgFromLine, 369
defun, 369
repeat, 400
syntax, 400
replaceFile, 975
defun, 975
replaceSharps, 948
calledby evaluateType1, 907
calls subCopy, 948
local ref $FormalMapVariableList, 948
local ref $TriangleVariableList, 948
defun, 948
reportInstantiations
calledby processInteractive, 50
reportOperations, 807
calledby showSpad2Cmd, 806
calls bright, 807
calls evaluateType, 808
calls isDomainValuedVariable, 807
calls isNameOfType, 807
calls isType, 808
calls mkAtree, 808
calls opOf, 808
calls qcar, 807
calls removeZeroOneDestructively, 808
calls reportOpsFromLisplib0, 808
calls reportOpsFromUnitDirectly0, 808
calls sayBrightly, 807
calls sayKeyedMsg, 807
calls unabbrev, 808
uses $doNotAddEmptyModeIfTrue, 808
uses $env, 808
uses $eval, 808
uses $genValue, 808
uses $quadSymbol, 808
defun, 807
reportOpsFromLisplib, 810
calledby reportOpsFromLisplib0, 809
calledby reportOpsFromLisplib1, 809
calls bright, 810
calls centerAndHighlight, 810
calls concat, 810
calls constructor?, 810
calls dc1, 810
calls displayOperationsFromLisplib, 810
calls eqsubstlist, 810
calls form2StringWithWhere, 810
calls form2String, 810
calls formatAttribute, 810
calls getConstructorSignature, 810
calls getdatabase, 810
calls isExposedConstructor, 810
calls kdr, 810
calls msort, 810
calls namestring, 810
calls nreverse0, 810
calls remdup, 810
calls say2PerLine, 810
calls sayBrightly, 810
calls sayKeyedMsg, 810
calls selectOptionLC, 810
calls specialChar, 810
calls strconc, 810
uses $FormalMapVariableList, 810
uses $linelength, 810
uses $options, 810
uses $showOptions, 810
defun, 810
reportOpsFromLisplib0, 809
calledby reportOperations, 808
calls reportOpsFromLisplib1, 809
calls reportOpsFromLisplib, 809
uses $useEditorForShowOutput, 809
defun, 809
reportOpsFromLisplib1, 809
calledby reportOpsFromLisplib0, 809

calls defiostream, 809
calls editFile, 809
calls erase, 809
calls pathname, 809
calls reportOpsFromLisplib, 809
calls sayShowWarning, 809
calls shut, 809
uses $erase, 809
uses $sayBrightlyStream, 809
defun, 809
reportOpsFromUnitDirectly, 813
calledby displayOperationsFromLisplib, 812
calledby reportOpsFromUnitDirectly0, 813
calledby reportOpsFromUnitDirectly1, 817
calls bright, 814
calls centerAndHighlight, 814
calls concat, 814
calls evalDomain, 814
calls formatAttribute, 814
calls formatOpType, 814
calls formatOperation, 814
calls getOplistForConstructorForm, 814
calls getdatabase, 814
calls getl, 814
calls isExposedConstructor, 814
calls member, 813
calls msort, 814
calls namestring, 814
calls nreverse0, 814
calls qcar, 814
calls remdup, 814
calls say2PerLine, 814
calls sayBrightly, 814
calls selectOptionLC, 814
calls specialChar, 814
calls strconc, 814
calls systemErrorHere, 814
uses $CategoryFrame, 814
uses $commentedOps, 814
uses $linelength, 814
uses $options, 814
uses $showOptions, 814
defun, 813
reportOpsFromUnitDirectly0, 813
calledby reportOperations, 808
calls reportOpsFromUnitDirectly1, 813
calls reportOpsFromUnitDirectly, 813
uses $useEditorForShowOutput, 813
defun, 813
reportOpsFromUnitDirectly1, 817
calledby reportOpsFromUnitDirectly0, 813
calls defiostream, 817
calls editFile, 817
calls erase, 817
calls pathname, 817
calls reportOpsFromUnitDirectly, 817
calls sayShowWarning, 817
calls shut, 817
uses $erase, 818
uses $sayBrightlyStream, 817
defun, 817
reportOpSymbol
calledby displayOperations, 533
reportSpadTrace, 882
calledby ?t, 893
calledby spadTrace,isTraceable, 866
calledby spadTrace, 867
calls qcar, 882
calls sayBrightly, 882
uses $traceNoisely, 882
defun, 882
reportUndo, 917
calledby diffAlist, 915
calls concat, 918
calls exit, 918
calls lassoc, 918
calls pname, 918
calls pp, 918
calls sayBrightlyNT, 918
calls sayBrightly, 918
calls seq, 918
uses $InteractiveFrame, 918
defun, 917
reportWhatOptions, 931
calledby whatSpad2Cmd, 930
calls sayBrightly, 931
uses $whatOptions, 931
defun, 931
reroot, 40
calledby initroot, 35
calls make-absolute-filename, 41

uses $current-directory, 41
uses $defaultMsgDatabaseName, 41
uses $directory-list, 41
uses $library-directory-list, 41
uses $msgDatabaseName, 41
uses $relative-directory-list, 41
uses $relative-library-directory-list, 41
uses $spadroot, 41
defun, 40
reset-stack-limits
calledby resetStackLimits, 21
resetCounters, 848
calledby trace1, 838
calls concat, 848
uses /countlist, 848
defun, 848
resethashtables, 991
calls browseopen, 991
calls categoryopen, 991
calls compressopen, 991
calls initial-getdatabase, 991
calls interpopen, 991
calls operationopen, 991
uses *allconstructors*, 992
uses *browse-stream*, 991
uses *category-stream*, 991
uses *category-stream-stamp*, 991
uses *compress-stream-stamp*, 992
uses *compressvector*, 992
uses *hascategory-hash*, 992
uses *interp-stream*, 991
uses *interp-stream-stamp*, 992
uses *operation-hash*, 992
uses *operation-stream*, 991
uses *operation-stream-stamp*, 992
uses *sourcefiles*, 991
defun, 991
resetInCoreHist, 584
calledby clearCmdAll, 499
calledby historySpad2Cmd, 578
uses $HistListAct, 584
uses $HistListLen, 584
uses $HistList, 584
defun, 584
resetSpacers, 848
calledby trace1, 838
calls concat, 848
uses /spacelist, 848
defun, 848
resetStackLimits, 21
calledby intloop, 26
calledby protectedEVAL, 47
calledby runspad, 21
calls reset-stack-limits, 21
defun, 21
resetTimers, 848
calledby trace1, 838
calls concat, 848
uses /timerlist, 848
defun, 848
resetWorkspaceVariables, 646
calls copy, 646
calls initializeSetVariables, 646
uses /countlist, 646
uses /editfile, 646
uses /pretty, 646
uses /sourcefiles, 646
uses /spacelist, 646
uses /timerlist, 646
uses $CommandSynonymAlist, 646
uses $IOindex, 646
uses $InitialCommandSynonymAlist, 646
uses $UserAbbreviationsAlist, 646
uses $boot, 646
uses $coerceIntByMapCounter, 646
uses $compileMapFlag, 646
uses $dependeeClosureAlist, 646
uses $echoLineStack, 646
uses $env, 646
uses $existingFiles, 646
uses $e, 646
uses $functionTable, 646
uses $msgAlist, 646
uses $msgDatabaseName, 646
uses $msgDatabase, 646
uses $operationNameList, 646
uses $setOptions, 646
uses $slamFlag, 646
uses $sourceFiles, 646
defun, 646
resolveTT
calledby unifyStructVar, 427

Rest, 77
calledby incLude1, 81
defmacro, 77
restart
calls get-current-directory, 17
calls init-memory-config, 17
calls initHist, 17
calls initializeInterpreterFrameRing, 17
calls initroot, 17
calls makeInitialModemapFrame, 17
calls openserver, 17
calls readSpadProfileIfThere, 17
calls restart0, 17
calls spadStartUpMsgs, 17
calls spad, 18
calls statisticsInitialization, 17
uses $IOindex, 18
uses $InteractiveFrame, 18
uses $SpadServername, 18
uses $SpadServer, 18
uses $current-directory, 18
uses $currentLine, 18
uses $displayStartMsgs, 18
uses $openServerIfTrue, 18
uses $printLoadMsgs, 18
restart function, 16
restart0, 18
calledby restart, 17
calls browseopen, 19
calls categoryopen, 19
calls compressopen, 18
calls getEnv, 19
calls interpopen, 19
calls operationopen, 19
defun, 18
restoreHistory, 593
calledby historySpad2Cmd, 578
calls $fcopy, 593
calls clearCmdSortedCaches, 593
calls clearSpad2Cmd, 593
calls disableHist, 593
calls get, 593
calls histFileErase, 593
calls histFileName, 593
calls identp, 593
calls makeHistFileName, 593
calls makeInputFilename, 593
calls namestring, 593
calls putHist, 593
calls qcar, 593
calls qcdr, 593
calls readHiFi, 593
calls rempropI, 593
calls rkeyids, 593
calls sayKeyedMsg, 593
calls throwKeyedMsg, 593
calls updateInCoreHist, 593
uses $HiFiAccess, 593
uses $InteractiveFrame, 593
uses $e, 593
uses $internalHistoryTable, 593
uses $oldHistoryFileName, 593
uses $options, 593
uses $useInternalHistoryTable, 593
defun, 593
retract, 1049
calledby printTypeAndTimeNormal, 59
calls isWrapped, 1049
calls objMode, 1049
calls objNew, 1049
calls objVal, 1049
calls qcar, 1049
calls retract1, 1049
local ref $EmptyMode, 1049
defun, 1049
retract1
calledby retract, 1049
rkeyids
calledby restoreHistory, 593
calledby setHistoryCore, 581
rplac
calledby spadTrace, 867
calledby spadUntrace, 886
rplacstr
calledby processSynonyms, 33
rread, 601
calledby SPADRREAD, 601
calledby rread, 601
calls rread, 601
defun, 601
rshut
calledby readHiFi, 597

calledby saveDependentsHashTable, 1013
calledby saveHistory, 592
calledby saveUsersHashTable, 1014
calledby setHistoryCore, 581
calledby writeHiFi, 598
ruleLhsTran, 320
calledby pfRule2Sex, 316
calls nsubst, 320
calls patternVarsOf, 320
uses $multiVarPredicateList, 320
uses $predicateList, 320
defun, 320
rulePredicateTran, 317
calledby pfRule2Sex, 316
calls patternVarsOf, 317
calls pvarPredTran, 317
uses $multiVarPredicateList, 317
defun, 317
runspad, 21
calledby spad, 20
calls exit, 21
calls ncTopLevel, 21
calls resetStackLimits, 21
calls seq, 21
uses $quitTag, 21
catches, 21
defun, 21
rwrite, 600
calledby rwrite, 600
calledby saveDependentsHashTable, 1013
calledby saveUsersHashTable, 1014
calledby spadrwrite0, 600
calls identp, 601
calls pname, 600, 601
calls rwrite, 600
defun, 600

safeWritify, 602
calledby spadrwrite0, 600
calls writify, 603
catches, 602
defun, 602
sameMsg?, 374
calledby redundant, 372
calls getMsgArgL, 374
calls getMsgKey, 374
defun, 374
sameUnionBranch, 61
calledby printTypeAndTimeNormal, 59
defun, 61
satisfiesRegularExpressions, 933
calledby filterListOfStringsWithFn, 933
calledby filterListOfStrings, 932
calls strpos, 933
defun, 933
satisfiesUserLevel, 447
calledby commandsForUserLevel, 444
calledby displaySetVariableSettings, 649
calledby set1, 800
uses $UserLevel, 447
defun, 447
save-system
calledby spad-save, 980
saveDependentsHashTable, 1013
calledby make-databases, 1010
calls erase, 1013
calls hget, 1013
calls hkeys, 1013
calls msort, 1013
calls rshut, 1013
calls rwrite, 1013
calls writeLib1, 1013
local ref $depTb, 1013
local ref $erase, 1013
defun, 1013
saveHistory, 591
calledby historySpad2Cmd, 578
calls histFileErase, 591
calls histFileName, 591
calls histInputFileName, 591
calls makeHistFileName, 591
calls makeInputFilename, 591
calls namestring, 592
calls object2Identifier, 592
calls rdefiostream, 591
calls rshut, 592
calls sayKeyedMsg, 591
calls spadrwrite0, 591
calls throwKeyedMsg, 591
calls writeInputLines, 591
uses $HiFiAccess, 592
uses $internalHistoryTable, 592

uses $seen, 592
uses $useInternalHistoryTable, 592
defun, 591
saveMapSig, 843
calledby trace1, 838
calls addassoc, 843
calls getMapSig, 843
calls rassoc, 843
uses $mapSubNameAlist, 843
uses $tracedMapSignatures, 843
defun, 843
savesystem, 642
calls helpSpad2Cmd, 642
calls spad-save, 642
defun, 642
savesystem help page, 641
manpage, 641
saveUsersHashTable, 1014
calledby make-databases, 1010
calls erase, 1014
calls hget, 1014
calls hkeys, 1014
calls msort, 1014
calls rshut, 1014
calls rwrite, 1014
calls writeLib1, 1014
local ref $erase, 1014
local ref $usersTb, 1014
defun, 1014
say
calledby displaySetVariableSettings, 649
calledby newHelpSpad2Cmd, 569
calledby trace1, 838
calledby whatCommands, 932
calledby workfilesSpad2Cmd, 939
say2PerLine
calledby displayOperationsFromLisplib, 813
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
sayAsManyPerLineAsPossible
calledby apropos, 935
calledby displayWorkspaceNames, 450
calledby setExposeAddGroup, 691
calledby setOutputCharacters, 758
calledby whatCommands, 932
sayBrightly
calledby ?t, 893
calledby break, 896
calledby describeFortPersistence, 748
calledby describeInputLibraryArgs, 660
calledby describeOutputLibraryArgs, 657
calledby describeSetFortDir, 718
calledby describeSetFortTmpDir, 716
calledby describeSetLinkerArgs, 720
calledby describeSetNagHost, 746
calledby describeSetOutputAlgebra, 756
calledby describeSetOutputFormula, 784
calledby describeSetOutputFortran, 763
calledby describeSetOutputHtml, 774
calledby describeSetOutputMathml, 770
calledby describeSetOutputOpenMath, 779
calledby describeSetOutputTex, 790
calledby displayCondition, 461
calledby displayMacros, 534
calledby displayMacro, 450
calledby displayModemap, 462
calledby displayMode, 462
calledby displaySetOptionInformation, 647
calledby displaySetVariableSettings, 649
calledby displayWorkspaceNames, 450
calledby msgOutputter, 351
calledby pcounters, 850
calledby printLabelledList, 470
calledby processKeyedError, 351
calledby pspacers, 849
calledby ptimers, 849
calledby reportOperations, 807
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calledby reportSpadTrace, 882
calledby reportUndo, 918
calledby reportWhatOptions, 931
calledby sayShowWarning, 818
calledby setFortDir, 717
calledby setFortTmpDir, 715
calledby setOutputCharacters, 758
calledby traceReply, 889
calledby workfilesSpad2Cmd, 940
calledby zsystemdevelopment1, 944
sayBrightly1, 1043
calledby sayMSG2File, 329

defun, 1043
saybrightly1
calledby saymsg, 329
calls brightprint-0, 1043
calls brightprint, 1043
sayBrightlyLength
calledby traceReply, 889
sayBrightlyNT
calledby letPrint, 875
calledby reportUndo, 918
sayExample, 535
defun, 535
sayexample
calls cleanupline, 536
calls sayNewLine, 536
sayKeyedMsg, 327
calledby abbQuery, 532
calledby abbreviationsSpad2Cmd, 479
calledby apropos, 935
calledby changeHistListLen, 585
calledby clearCmdAll, 499
calledby clearCmdCompletely, 498
calledby clearCmdParts, 501
calledby clearSpad2Cmd, 496
calledby commandAmbiguityError, 448
calledby commandErrorMessage, 446
calledby describeSetStreamsCalculate, 793
calledby displayExposedConstructors, 696
calledby displayExposedGroups, 696
calledby displayFrameNames, 559
calledby displayHiddenConstructors, 697
calledby displayOperations, 533
calledby displayProperties, 457
calledby handleNoParseCommands, 466
calledby helpSpad2Cmd, 568
calledby historySpad2Cmd, 578
calledby history, 578
calledby importFromFrame, 559
calledby listConstructorAbbreviations, 481
calledby loadLibNoUpdate, 1031
calledby loadLib, 1029
calledby load, 625
calledby localdatabase, 1005
calledby localnrlib, 1007
calledby newHelpSpad2Cmd, 569
calledby npsystem, 468
calledby printStatisticsSummary, 57
calledby printTypeAndTimeNormal, 59
calledby quitSpad2Cmd, 634
calledby reportOperations, 807
calledby reportOpsFromLisplib, 810
calledby restoreHistory, 593
calledby saveHistory, 591
calledby set1, 800
calledby setExposeAddConstr, 692
calledby setExposeAddGroup, 691
calledby setExposeAdd, 690
calledby setExposeDropConstr, 695
calledby setExposeDropGroup, 694
calledby setExposeDrop, 693
calledby setExpose, 689
calledby setHistoryCore, 581
calledby setOutputAlgebra, 754
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby showSpad2Cmd, 806
calledby spadStartUpMsgs, 19
calledby trace1, 838
calledby undoInCore, 589
calledby userLevelErrorMessage, 447
calledby whatCommands, 932
calledby whatSpad2Cmd, 930
calledby workfilesSpad2Cmd, 939
calledby writeInputLines, 583
calledby writifyComplain, 602
calls sayKeyedMsgLocal, 327
uses $texFormatting, 328
defun, 327
sayKeyedMsgLocal, 328
calledby sayKeyedMsg, 327
calls flowSegmentedMsg, 328
calls getKeyedMsg, 328
calls sayMSG2File, 328
calls sayMSG, 328
calls segmentKeyedMsg, 328
calls substituteSegmentedMsg, 328
uses $displayMsgNumber, 328
uses $linelength, 328

uses $margin, 328
uses $printMsgsToFile, 328
defun, 328
sayMessage
calledby apropos, 935
calledby clearCmdParts, 501
calledby describeSpad2Cmd, 525
calledby displaySetOptionInformation, 647
calledby displaySpad2Cmd, 532
calledby displayWorkspaceNames, 450
calledby filterAndFormatConstructors, 934
calledby printLabelledList, 470
calledby set1, 800
calledby setFortPers, 748
calledby setOutputCharacters, 758
calledby setStreamsCalculate, 793
calledby traceReply, 889
calledby updateFromCurrentInterpreter-Frame, 554
calledby whatCommands, 932
calledby zsystemdevelopment1, 944
sayMSG, 329
calledby ?t, 893
calledby commandAmbiguityError, 448
calledby displayProperties,sayFunctionDeps, 454
calledby displayProperties, 458
calledby displaySetOptionInformation, 647
calledby displayType, 456
calledby displayValue, 455
calledby evalDomain, 903
calledby getAndSay, 457
calledby initializeSetVariables, 645
calledby sayKeyedMsgLocal, 328
calledby set1, 800
calledby setExposeAddGroup, 690
calledby setExposeAdd, 689
calledby setExposeDropConstr, 695
calledby setExposeDropGroup, 694
calledby setExposeDrop, 693
calledby setExpose, 689
calledby showInOut, 596
calledby showInput, 595
calledby spadStartUpMsgs, 19
calledby spadUntrace, 885
calledby traceDomainLocalOps, 870
calledby traceReply, 889
calledby untraceDomainLocalOps, 870
uses $algebraOutputStream, 329
defun, 329
saymsg
calls saybrightly1, 329
sayMSG2File, 329
calledby sayKeyedMsgLocal, 328
calls defiostream, 329
calls makePathname, 329
calls sayBrightly1, 329
calls shut, 329
defun, 329
sayNewLine
calledby sayexample, 536
sayShowWarning, 818
calledby reportOpsFromLisplib1, 809
calledby reportOpsFromUnitDirectly1, 817
calls sayBrightly, 818
defun, 818
scanCheckRadix, 131
calledby scanNumber, 129
uses $linepos, 132
uses $n, 132
defun, 131
scanCloser, 122
usedby scanCloser?, 123
defvar, 122
scanCloser?, 123
calledby scanKeyTr, 117
calls keyword, 123
uses scanCloser, 123
defun, 123
scanComment, 113
calledby scanToken, 111
calls lfcomment, 113
calls substring, 113
uses $ln, 113
uses $n, 113
uses $sz, 113
defun, 113
scanDict, 134
defvar, 134
scanDictCons, 134
calls hkeys, 135
defun, 134

scanError, 132
calledby scanPunct, 115
calledby scanToken, 112
calls lferror, 133
calls lnExtraBlanks, 132
calls ncSoftError, 132
uses $linepos, 133
uses $ln, 133
uses $n, 133
defun, 132
scanEsc, 120
calledby scanEscape, 132
calledby scanEsc, 121
calledby scanS, 128
calledby scanW, 125
calledby spleI1, 120
calls nextline, 121
calls qenum, 121
calls scanEsc, 121
calls startsComment?, 121
calls startsNegComment?, 121
calls strposl, 121
uses $ln, 121
uses $n, 121
uses $r, 121
uses $sz, 121
defun, 120
scanEscape, 132
calledby scanToken, 112
calls scanEsc, 132
calls scanWord, 132
uses $n, 132
defun, 132
scanExponent, 124
calledby scanNumber, 129
calledby scanPossFloat, 118
calls concat, 124
calls digit?, 124
calls lffloat, 124
calls qenum, 124
calls spleI, 124
uses $ln, 124
uses $n, 124
uses $sz, 124
defun, 124
scanIgnoreLine, 110
calledby lineoftoks, 109
calls incPrefix?, 110
calls qenum, 110
defun, 110
scanInsert, 135
calls qenum, 136
defun, 135
scanKeyTable, 133
defvar, 133
scanKeyTableCons, 133
defun, 133
scanKeyTr, 117
calledby scanPunct, 115
calls keyword, 117
calls lfkey, 117
calls scanCloser?, 117
calls scanPossFloat, 117
uses $floatok, 117
defun, 117
scanKeyWords, 105
defvar, 105
scanNegComment, 114
calledby scanToken, 111
calls lfnegcomment, 114
calls substring, 114
uses $ln, 114
uses $n, 115
uses $sz, 114
defun, 114
scanNumber, 129
calledby scanToken, 111
calls concat, 129
calls lfinteger, 129
calls lfrinteger, 129
calls qenum, 129
calls scanCheckRadix, 129
calls scanExponent, 129
calls spleI1, 129
calls spleI, 129
uses $floatok, 129
uses $ln, 130
uses $n, 130
uses $sz, 130
defun, 129
ScanOrPairVec, 612
calledby dewritify, 611

calledby writify, 606
calls ScanOrPairVec,ScanOrInner, 612
uses $seen, 612
catches, 612
defun, 612
ScanOrPairVec,ScanOrInner, 611
calledby ScanOrPairVec,ScanOrInner, 611
calledby ScanOrPairVec, 612
calls ScanOrPairVec,ScanOrInner, 611
calls hget, 611
calls hput, 611
calls qcar, 611
calls qcdr, 611
calls vecp, 611
uses $seen, 611
defun, 611
throws, 611
scanPossFloat, 118
calledby scanKeyTr, 117
calls digit?, 118
calls lfkey, 118
calls scanExponent, 118
calls spleI, 118
uses $ln, 118
uses $n, 119
uses $sz, 118
defun, 118
scanPun, 136
defvar, 136
scanPunCons, 137
calls hkeys, 137
defun, 137
scanPunct, 115
calledby scanToken, 111
calls scanError, 115
calls scanKeyTr, 115
calls subMatch, 115
uses $ln, 115
uses $n, 115
defun, 115
scanS, 128
calledby scanString, 127
calledby scanS, 128
calls concat, 128
calls lnExtraBlanks, 128
calls ncSoftError, 128
calls scanEsc, 128
calls scanS, 128
calls scanTransform, 128
calls strpos, 128
calls substring, 128
uses $linepos, 128
uses $ln, 128
uses $n, 128
uses $sz, 128
defun, 128
scanSpace, 127
calledby scanToken, 111
calls lfspaces, 127
calls strposl, 127
uses $floatok, 127
uses $ln, 127
uses $n, 127
defun, 127
scanString, 127
calledby scanToken, 111
calls lfstring, 127
calls scanS, 127
uses $floatok, 127
uses $n, 127
defun, 127
scanToken, 111
calls constoken, 112
calls digit?, 111
calls dqUnit, 112
calls lfid, 111
calls lnExtraBlanks, 112
calls punctuation?, 111
calls qenum, 111
calls scanComment, 111
calls scanError, 112
calls scanEscape, 112
calls scanNegComment, 111
calls scanNumber, 111
calls scanPunct, 111
calls scanSpace, 111
calls scanString, 111
calls scanWord, 111
calls startsComment?, 111
calls startsId?, 111
calls startsNegComment?, 111
uses $linepos, 112

uses $ln, 112
uses $n, 112
defun, 111
scanTransform, 129
calledby scanS, 128
defun, 129
scanW, 125
calledby scanWord, 123
calledby scanW, 125
calls concat, 125
calls idChar?, 125
calls posend, 125
calls qenum, 125
calls scanEsc, 125
calls scanW, 125
calls substring, 125
uses $ln, 125
uses $n, 125
uses $sz, 125
defun, 125
scanWord, 123
calledby scanEscape, 132
calledby scanToken, 111
calls keyword?, 123
calls lfid, 123
calls lfkey, 123
calls scanW, 123
uses $floatok, 123
defun, 123
search, 954
calledby getProplist, 954
calls searchCurrentEnv, 954
calls searchTailEnv, 954
defun, 954
searchCurrentEnv, 954
calledby search, 954
calls assq, 954
calls kdr, 954
defun, 954
searchTailEnv, 955
calledby search, 954
calls assq, 955
calls kdr, 955
defun, 955
sec, 1089
defun, 1089
sech, 1091
defun, 1091
segmentKeyedMsg, 328
calledby getMsgInfoFromKey, 364
calledby msgText, 61
calledby sayKeyedMsgLocal, 328
calls string2Words, 328
defun, 328
selectOption, 475
calledby selectOptionLC, 475
calledby set1, 800
calledby systemCommand, 444
calledby unAbbreviateKeyword, 465
calls identp, 475
calls member, 475
calls pname, 475
calls qcar, 476
calls qcdr, 475
calls stringPrefix?, 475
defun, 475
selectOptionLC, 475
calledby abbreviationsSpad2Cmd, 479
calledby clearCmdParts, 501
calledby clearSpad2Cmd, 496
calledby close, 506
calledby describeSpad2Cmd, 525
calledby displaySpad2Cmd, 532
calledby frameSpad2Cmd, 562
calledby getTraceOption, 844
calledby historySpad2Cmd, 578
calledby newHelpSpad2Cmd, 569
calledby readSpad2Cmd, 638
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calledby setExposeAdd, 690
calledby setExposeDrop, 693
calledby setExpose, 689
calledby setInputLibrary, 659
calledby synonymsForUserLevel, 825
calledby systemCommand, 444
calledby trace1, 838
calledby unAbbreviateKeyword, 465
calledby whatSpad2Cmd, 930
calledby workfilesSpad2Cmd, 939
calledby zsystemdevelopment1, 944
calls downcase, 475

calls object2Identifier, 475
calls selectOption, 475
defun, 475
sendHTErrorSignal
calledby protectedEVAL, 47
separatePiles, 339
calledby pileCforest, 338
calledby separatePiles, 339
calls dqConcat, 339
calls dqUnit, 339
calls lastTokPosn, 339
calls separatePiles, 339
calls tokConstruct, 339
defun, 339
SEQ
calledby funfind,LAM, 864
seq
calledby /tracereply, 884
calledby abbreviationsSpad2Cmd, 479
calledby apropos, 935
calledby clearCmdParts, 501
calledby coerceSpadArgs2E, 855
calledby dewritify,dewritifyInner, 608
calledby diffAlist, 915
calledby displayMacros, 534
calledby displayProperties,sayFunctionDeps, 454
calledby displayProperties, 458
calledby flattenOperationAlist, 873
calledby getAliasIfTracedMapParameter, 880
calledby getBpiNameIfTracedMap, 880
calledby getPreviousMapSubNames, 860
calledby getTraceOption,hn, 843
calledby getTraceOptions, 842
calledby getTraceOption, 844
calledby getWorkspaceNames, 451
calledby historySpad2Cmd, 579
calledby importFromFrame, 559
calledby isListOfIdentifiersOrStrings, 859
calledby isListOfIdentifiers, 858
calledby orderBySlotNumber, 883
calledby prTraceNames,fn, 888
calledby prTraceNames, 888
calledby recordFrame, 913
calledby removeUndoLines, 923
calledby reportUndo, 918
calledby runspad, 21
calledby set1, 800
calledby spadReply,printName, 884
calledby spadReply, 885
calledby spadTrace,isTraceable, 865
calledby spadTrace, 866
calledby spadUntrace, 886
calledby subTypes, 856
calledby trace1, 838
calledby traceDomainConstructor, 870
calledby traceReply, 889
calledby undoFromFile, 590
calledby undoLocalModemapHack, 923
calledby undoSingleStep, 921
calledby untraceDomainConstructor,keepTraced?, 872
calledby untraceDomainConstructor, 873
calledby whatConstructors, 935
calledby whatSpad2Cmd, 930
calledby writify,writifyInner, 603
serverReadLine, 44
calledby intloopReadConsole, 30
calls addNewInterpreterFrame, 44
calls changeToNamedInterpreterFrame, 44
calls executeQuietCommand, 44
calls is-console[9], 45
calls lassoc, 44
calls mkprompt, 44
calls parseAndInterpret, 44
calls protectedEVAL, 44
calls read-line, 44
calls serverSwitch, 45
calls sockGetInt, 44
calls sockGetString, 44
calls sockSendInt, 44
calls sockSendString, 44
calls unescapeStringsInForm, 44
uses *eof*, 45
uses $CallInterp, 45
uses $CreateFrameAnswer, 45
uses $CreateFrame, 45
uses $EndOfOutput, 45
uses $EndServerSession, 45
uses $EndSession, 45
uses $KillLispSystem, 45

uses $LispCommand, 45
uses $MenuServer, 45
uses $NeedToSignalSessionManager, 45
uses $NonSmanSession, 45
uses $QuietSpadCommand, 45
uses $SessionManager, 45
uses $SpadCommand, 45
uses $SpadServer, 45
uses $SwitchFrames, 45
uses $currentFrameNum, 45
uses $frameAlist, 45
uses $frameNumber, 45
uses $sockBufferLength, 45
uses in-stream, 45
catches, 44
defun, 44
serverSwitch
calledby serverReadLine, 45
set, 799
calledby browse, 489
calls set1, 799
uses $setOptions, 799
defun, 799
set help page, 643
manpage, 643
set-hole-size
calledby init-memory-config, 34
set-restart-hook, 15
defun, 15
set1, 800
calledby set1, 801
calledby set, 799
calls bright, 800
calls displaySetOptionInformation, 800
calls displaySetVariableSettings, 800
calls downcase, 800
calls exit, 800
calls kdr, 800
calls lassoc, 800
calls literals, 801
calls object2String, 800
calls poundsign, 800
calls satisfiesUserLevel, 800
calls sayKeyedMsg, 800
calls sayMSG, 800
calls sayMessage, 800
calls selectOption, 800
calls seq, 800
calls set1, 801
calls translateYesNo2TrueFalse, 801
calls tree, 801
calls use-fast-links, 801
uses $UserLevel, 801
uses $displaySetValue, 801
uses $setOptionNames, 801
defun, 800
setCurrentLine, 41
calledby intloopEchoParse, 69
calledby intloopProcessString, 37
calledby intloopProcess, 64
calledby intloopReadConsole, 30
uses $currentLine, 42
defun, 41
setdatabase, 1000
calledby abbreviationsSpad2Cmd, 479
calls make-database, 1000
defun, 1000
setdifference
calledby displayWorkspaceNames, 450
calledby untraceMapSubNames, 863
setelt
calledby setExposeAddConstr, 692
calledby setExposeAddGroup, 690
calledby setExposeDropConstr, 695
calledby setExposeDropGroup, 694
setelt32, 1048
defmacro, 1048
setExpose, 688
calledby setExpose, 689
calls displayExposedConstructors, 689
calls displayExposedGroups, 688
calls displayHiddenConstructors, 689
calls namestring, 689
calls pathname, 689
calls qcar, 689
calls qcdr, 689
calls sayKeyedMsg, 689
calls sayMSG, 689
calls selectOptionLC, 689
calls setExposeAdd, 689
calls setExposeDrop, 689
calls setExpose, 689

defun, 688
setExposeAdd, 689
calledby setExposeAdd, 690
calledby setExpose, 689
calls centerAndHighlight, 689
calls displayExposedConstructors, 689
calls displayExposedGroups, 689
calls qcar, 690
calls qcdr, 690
calls sayKeyedMsg, 690
calls sayMSG, 689
calls selectOptionLC, 690
calls setExposeAddConstr, 690
calls setExposeAddGroup, 690
calls setExposeAdd, 690
calls specialChar, 689
uses $linelength, 690
defun, 689
setExposeAddConstr, 692
calledby localnrlib, 1007
calledby setExposeAdd, 690
calls centerAndHighlight, 692
calls clearClams, 692
calls delete, 692
calls displayExposedConstructors, 692
calls getdatabase, 692
calls member, 692
calls msort, 692
calls qcar, 692
calls sayKeyedMsg, 692
calls setelt, 692
calls specialChar, 692
calls unabbrev, 692
uses $interpreterFrameName, 692
uses $linelength, 692
uses $localExposureData, 692
defun, 692
setExposeAddGroup, 690
calledby setExposeAdd, 690
calls centerAndHighlight, 691
calls clearClams, 690
calls displayExposedConstructors, 690
calls displayExposedGroups, 690
calls displayHiddenConstructors, 690
calls getalist, 691
calls member, 691
calls msort, 691
calls namestring, 691
calls object2String, 690
calls pathname, 691
calls qcar, 690
calls sayAsManyPerLineAsPossible, 691
calls sayKeyedMsg, 691
calls sayMSG, 690
calls setelt, 690
calls specialChar, 691
uses $globalExposureGroupAlist, 691
uses $interpreterFrameName, 691
uses $linelength, 691
uses $localExposureData, 691
defun, 690
setExposeDrop, 693
calledby setExposeDrop, 693
calledby setExpose, 689
calls centerAndHighlight, 693
calls displayHiddenConstructors, 693
calls qcar, 693
calls qcdr, 693
calls sayKeyedMsg, 693
calls sayMSG, 693
calls selectOptionLC, 693
calls setExposeDropConstr, 693
calls setExposeDropGroup, 693
calls setExposeDrop, 693
calls specialChar, 693
uses $linelength, 693
defun, 693
setExposeDropConstr, 695
calledby setExposeDrop, 693
calls centerAndHighlight, 695
calls clearClams, 695
calls delete, 695
calls displayExposedConstructors, 695
calls displayHiddenConstructors, 695
calls getdatabase, 695
calls member, 695
calls msort, 695
calls qcar, 695
calls sayKeyedMsg, 695
calls sayMSG, 695
calls setelt, 695
calls specialChar, 695

calls unabbrev, 695
uses $interpreterFrameName, 695
uses $linelength, 695
uses $localExposureData, 695
defun, 695
setExposeDropGroup, 694
calledby setExposeDrop, 693
calls centerAndHighlight, 694
calls clearClams, 694
calls delete, 694
calls displayExposedConstructors, 694
calls displayExposedGroups, 694
calls displayHiddenConstructors, 694
calls getalist, 694
calls member, 694
calls qcar, 694
calls sayKeyedMsg, 694
calls sayMSG, 694
calls setelt, 694
calls specialChar, 694
uses $globalExposureGroupAlist, 694
uses $interpreterFrameName, 694
uses $linelength, 694
uses $localExposureData, 694
defun, 694
setFortDir, 717
calls bright, 718
calls describeSetFortDir, 717
calls pname, 717
calls sayBrightly, 717
calls validateOutputDirectory, 717
uses $fortranDirectory, 718
defun, 717
setFortPers, 748
calls bright, 748
calls describeFortPersistence, 748
calls sayMessage, 748
calls terminateSystemCommand, 748
uses $fortPersistence, 748
defun, 748
setFortTmpDir, 715
calls bright, 715
calls describeSetFortTmpDir, 715
calls pname, 715
calls sayBrightly, 715
calls validateOutputDirectory, 715
uses $fortranTmpDir, 715
defun, 715
setFunctionsCache, 699
defun, 699
setHistoryCore, 580
calledby historySpad2Cmd, 578
calls boot-equal, 581
calls disableHist, 581
calls histFileErase, 581
calls histFileName, 581
calls object2Identifier, 581
calls rdefiostream, 581
calls readHiFi, 581
calls rkeyids, 581
calls rshut, 581
calls sayKeyedMsg, 581
calls spadrwrite, 581
uses $HiFiAccess, 581
uses $IOindex, 581
uses $internalHistoryTable, 581
uses $useInternalHistoryTable, 581
defun, 580
setInputLibrary, 659
calledby setInputLibrary, 659
calls addInputLibrary, 659
calls describeInputLibraryArgs, 659
calls dropInputLibrary, 659
calls qcar, 659
calls qcdr, 659
calls selectOptionLC, 659
calls setInputLibrary, 659
uses input-libraries, 659
defun, 659
setIOindex, 595
uses $IOindex, 595
defun, 595
setletprintflag
calledby breaklet, 895
calledby spadTrace, 867
calledby spadUntrace, 886
calledby tracelet, 894
setLinkerArgs, 720
calls describeSetLinkerArgs, 720
calls object2String, 720
uses $fortranLibraries, 720
defun, 720

setMsgCatlessAttr, 363
calledby setMsgForcedAttr, 362
calledby setMsgUnforcedAttr, 365
calls ifcdr, 363
calls ncAlist, 363
calls ncPutQ, 363
calls qassq, 363
defun, 363
setMsgForcedAttr, 362
calledby setMsgForcedAttrList, 362
calls ncPutQ, 362
calls setMsgCatlessAttr, 362
defun, 362
setMsgForcedAttrList, 362
calledby msgCreate, 346
calls setMsgForcedAttr, 362
calls whichCat, 362
defun, 362
setMsgPrefix, 347
calledby processChPosesForOneLine, 374
defun, 347
setMsgText, 347
calledby putDatabaseStuff, 363
calledby putFTText, 377
defun, 347
setMsgUnforcedAttr, 365
calledby initImPr, 365
calledby initToWhere, 366
calledby setMsgUnforcedAttrList, 364
calls ncAlist, 365
calls ncPutQ, 365
calls qassq, 365
calls setMsgCatlessAttr, 365
defun, 365
setMsgUnforcedAttrList, 364
calledby putDatabaseStuff, 363
calls setMsgUnforcedAttr, 364
calls whichCat, 364
defun, 364
setNagHost, 746
calls describeSetNagHost, 746
calls object2String, 746
uses $nagHost, 746
defun, 746
setOutputAlgebra, 754
calledby describeSetOutputAlgebra, 756
calledby spad, 20
calls $filep, 754
calls concat, 754
calls defiostream, 754
calls describeSetOutputAlgebra, 754
calls make-outstream, 754
calls member, 754
calls object2String, 754
calls pathnameDirectory, 754
calls pathnameName, 754
calls pathnameType, 754
calls qcar, 754
calls qcdr, 754
calls sayKeyedMsg, 754
calls shut, 754
calls upcase, 754
uses $algebraFormat, 754
uses $algebraOutputFile, 754
uses $algebraOutputStream, 754
uses $filep, 754
defun, 754
setOutputCharacters, 758
calledby setOutputCharacters, 758
calls bright, 758
calls concat, 758
calls downcase, 758
calls pname, 758
calls qcar, 758
calls qcdr, 758
calls sayAsManyPerLineAsPossible, 758
calls sayBrightly, 758
calls sayMessage, 758
calls setOutputCharacters, 758
calls specialChar, 758
uses $RTspecialCharacters, 758
uses $plainRTspecialCharacters, 758
uses $specialCharacterAlist, 758
uses $specialCharacters, 758
defun, 758
setOutputFormula, 781
calledby describeSetOutputFormula, 784
calls $filep, 782
calls concat, 782
calls defiostream, 781
calls describeSetOutputFormula, 782
calls make-outstream, 782

calls member, 782
calls object2String, 782
calls pathnameDirectory, 782
calls pathnameName, 782
calls pathnameType, 782
calls qcar, 782
calls qcdr, 782
calls sayKeyedMsg, 782
calls shut, 782
calls upcase, 782
uses $filep, 782
uses $formulaFormat, 782
uses $formulaOutputFile, 782
uses $formulaOutputStream, 782
defun, 781
setOutputFortran, 761
calledby describeSetOutputFortran, 763
calls $filep, 761
calls concat, 761
calls defiostream, 761
calls describeSetOutputFortran, 761
calls makeStream, 761
calls member, 761
calls object2String, 761
calls pathnameDirectory, 761
calls pathnameName, 761
calls pathnameType, 761
calls qcar, 761
calls qcdr, 761
calls sayKeyedMsg, 761
calls shut, 761
calls upcase, 761
uses $filep, 761
uses $fortranFormat, 761
uses $fortranOutputFile, 761
uses $fortranOutputStream, 761
defun, 761
setOutputHtml, 772
calledby describeSetOutputHtml, 774
calls $filep, 772
calls concat, 772
calls defiostream, 772
calls describeSetOutputHtml, 772
calls make-outstream, 772
calls member, 772
calls object2String, 772
calls pathnameDirectory, 772
calls pathnameName, 772
calls pathnameType, 772
calls qcar, 772
calls qcdr, 772
calls sayKeyedMsg, 772
calls shut, 772
calls upcase, 772
uses $filep, 773
uses $htmlFormat, 773
uses $htmlOutputFile, 773
uses $htmlOutputStream, 772
defun, 772
setOutputLibrary, 656
calls describeOutputLibraryArgs, 656
calls filep, 656
calls openOutputLibrary, 656
calls poundsign, 656
uses $outputLibraryName, 656
defun, 656
setOutputMathml, 768
calledby describeSetOutputMathml, 770
calls $filep, 768
calls concat, 768
calls defiostream, 768
calls describeSetOutputMathml, 768
calls make-outstream, 768
calls member, 768
calls object2String, 768
calls pathnameDirectory, 768
calls pathnameName, 768
calls pathnameType, 768
calls qcar, 768
calls qcdr, 768
calls sayKeyedMsg, 768
calls shut, 768
calls upcase, 768
uses $filep, 768
uses $mathmlFormat, 768
uses $mathmlOutputFile, 768
uses $mathmlOutputStream, 768
defun, 768
setOutputOpenMath, 777
calledby describeSetOutputOpenMath, 779
calls $filep, 777
calls concat, 777

calls defiostream, 777
calls describeSetOutputOpenMath, 777
calls make-outstream, 777
calls member, 777
calls object2String, 777
calls pathnameDirectory, 777
calls pathnameName, 777
calls pathnameType, 777
calls qcar, 777
calls qcdr, 777
calls sayKeyedMsg, 777
calls shut, 777
calls upcase, 777
uses $filep, 777
uses $openMathFormat, 777
uses $openMathOutputFile, 777
uses $openMathOutputStream, 777
defun, 777
setOutputTex, 788
calledby describeSetOutputTex, 790
calls $filep, 788
calls concat, 788
calls defiostream, 788
calls describeSetOutputTex, 788
calls make-outstream, 788
calls member, 788
calls object2String, 788
calls pathnameDirectory, 788
calls pathnameName, 788
calls pathnameType, 788
calls qcar, 788
calls qcdr, 788
calls sayKeyedMsg, 788
calls shut, 788
calls upcase, 788
uses $filep, 788
uses $texFormat, 788
uses $texOutputFile, 788
uses $texOutputStream, 788
defun, 788
setStreamsCalculate, 793
calls bright, 793
calls describeSetStreamsCalculate, 793
calls object2String, 793
calls sayMessage, 793
calls terminateSystemCommand, 793
uses $streamCount, 793
defun, 793
shortenForPrinting, 881
calledby letPrint, 875
calls devaluate, 881
calls isDomainOrPackage, 881
defun, 881
show, 806
calls showSpad2Cmd, 806
defun, 806
show help page, 805
manpage, 805
showdatabase, 999
calls getdatabase, 999
defun, 999
showHistory
calledby historySpad2Cmd, 578
showInOut, 596
calls assq, 596
calls disableHist, 596
calls objMode, 596
calls objValUnwrap, 596
calls readHiFi, 596
calls sayMSG, 596
calls spadPrint, 596
defun, 596
showInput, 595
calls disableHist, 595
calls readHiFi, 595
calls sayMSG, 595
calls tab, 595
defun, 595
showMsgPos?, 355
calledby getPosStL, 354
calls leader?, 356
calls msgImPr?, 355
uses $erMsgToss, 356
defun, 355
showSpad2Cmd, 806
calledby show, 806
calls helpSpad2Cmd, 806
calls member, 806
calls qcar, 806
calls reportOperations, 806
calls sayKeyedMsg, 806
uses $InteractiveFrame, 807

uses $env, 807
uses $e, 807
uses $options, 807
uses $showOptions, 807
defun, 806
shut, 972
calledby reportOpsFromLisplib1, 809
calledby reportOpsFromUnitDirectly1, 817
calledby sayMSG2File, 329
calledby setOutputAlgebra, 754
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby writeInputLines, 583
calledby zsystemdevelopment1, 944
calls is-console[9], 972
defun, 972
size, 1039
calledby abbreviationsSpad2Cmd, 479
calledby getPreStL, 353
calledby isgenvar, 876
calledby makeMsgFromLine, 369
calledby processChPosesForOneLine, 374
calledby processSynonyms, 33
calledby substringMatch, 117
calledby writeInputLines, 583
defun, 1039
SkipEnd?, 80
calledby incLude1, 82
calls remainder, 80
defvar, 80
SkipPart?, 81
calledby incLude1, 82
calls remainder, 81
defvar, 81
Skipping?, 81
calledby incLude1, 81
calls KeepPart?, 81
defvar, 81
sockGetInt
calledby queryClients, 506
calledby serverReadLine, 44
sockGetString
calledby executeQuietCommand, 47
calledby serverReadLine, 44
sockSendInt
calledby close, 506
calledby queryClients, 506
calledby serverReadLine, 44
sockSendString
calledby serverReadLine, 44
sortby
calledby workfilesSpad2Cmd, 939
spad, 20
calledby restart, 18
calls runspad, 20
calls setOutputAlgebra, 20
uses $PrintCompilerMessageIfTrue, 20
defun, 20
spad-error-loc, 960
calledby spad-long-error, 959
defun, 960
spad-long-error, 959
calledby spad-syntax-error, 959
calls iostat, 959
calls spad-error-loc, 959
uses out-stream, 959
uses spaderrorstream, 959
defun, 959
spad-save, 979
calledby savesystem, 642
calls save-system, 980
uses $SpadServer, 980
uses $openServerIfTrue, 980
defun, 979
spad-short-error, 960
calledby spad-syntax-error, 959
calls line-past-end-p, 960
calls line-print, 960
uses $current-line, 960
defun, 960
spad-syntax-error, 959
calls bumperrorcount, 959
calls consoleinputp, 959
calls ioclear, 959
calls spad-long-error, 959
calls spad-short-error, 959
uses debugmode, 959
defun, 959

throws, 959
spad2BootCoerce, 1051
defun, 1051
spadcall
calledby basicLookupCheckDefaults, 1063
calledby basicLookup, 1061
calledby clearCmdSortedCaches, 497
calledby letPrint3, 878
calledby lookupInDomainVector, 1063
spadClosure?, 607
calledby writify,writifyInner, 603
calls bpiname, 607
calls qcar, 607
calls qcdr, 607
calls vecp, 607
defun, 607
spadConstant, 1139
defmacro, 1139
spaddifference
calledby changeHistListLen, 585
calledby charDigitVal, 613
calledby dewritify,dewritifyInner, 608
calledby displaySetVariableSettings, 649
calledby fetchOutput, 596
calledby getAliasIfTracedMapParameter, 879
calledby removeUndoLines, 924
calledby undoCount, 919
calledby undoInCore, 588
calledby undoSteps, 920
calledby undo, 912
calledby writeInputLines, 583
spaderrorstream
usedby init-boot/spad-reader, 958
usedby spad-long-error, 959
SpadInterpretStream, 27
calledby intloop, 26
calls intloopInclude, 29
calls intloopReadConsole, 29
calls mkprompt, 29
uses $erMsgToss, 29
uses $fn, 29
uses $inclAssertions, 29
uses $lastPos, 29
uses $libQuiet, 29
uses $ncMsgList, 29
uses $newcompErrorCount, 29
uses $nopos, 29
uses $okToExecuteMachineCode, 29
uses $promptMsg, 29
uses $systemCommandFunction, 29
defun, 27
spadPrint
calledby showInOut, 596
spadReply, 885
calledby spadTrace, 867
calledby spadUntrace, 886
calls exit, 885
calls seq, 885
calls spadReply,printName, 885
uses /tracenames, 885
defun, 885
spadReply,printName, 884
calledby spadReply, 885
calls devaluate, 884
calls exit, 884
calls isDomainOrPackage, 884
calls qcar, 884
calls seq, 884
defun, 884
SPADRREAD
calls dewritify, 601
calls rread, 601
spadrread, 601
calledby readHiFi, 597
defun, 601
spadrwrite, 601
calledby setHistoryCore, 581
calledby writeHiFi, 598
calls spadrwrite0, 601
calls throwKeyedMsg, 601
defun, 601
spadrwrite0, 600
calledby saveHistory, 591
calledby spadrwrite, 601
calls rwrite, 600
calls safeWritify, 600
defun, 600
spadStartUpMsgs, 19
calledby restart, 17
calls fillerSpaces, 19
calls sayKeyedMsg, 19

calls sayMSG, 19
calls specialChar, 19
uses *build-version*, 19
uses *yearweek*, 19
uses $linelength, 19
uses $msgAlist, 19
uses $opSysName, 19
defun, 19
spadsysnamep
calledby coerceTraceArgs2E, 854
calledby coerceTraceFunValue2E, 857
spadThrow
calledby pf2Sex1, 299
calledby tersyscommand, 448
calledby throwEvalTypeMsg, 909
spadTrace, 866
calledby traceDomainConstructor, 871
calls aldorTrace, 866
calls as-insert, 866
calls assoc, 866
calls bpiname, 867
calls bpitrace, 867
calls constructSubst, 867
calls exit, 866
calls flattenOperationAlist, 866
calls getOperationAlistFromLisplib, 866
calls getOption, 866
calls isDomainOrPackage, 866
calls kdr, 866
calls opOf, 866
calls printDashedLine, 867
calls refvecp, 866
calls removeOption, 866
calls reportSpadTrace, 867
calls rplac, 867
calls seq, 866
calls setletprintflag, 867
calls spadReply, 867
calls spadTrace,g, 866
calls spadTrace,isTraceable, 866
calls spadTraceAlias, 867
calls subTypes, 867
calls userError, 866
uses /tracenames, 867
uses $fromSpadTrace, 867
uses $letAssoc, 867
uses $reportSpadTrace, 867
uses $traceNoisely, 867
uses $tracedModemap, 867
defun, 866
spadTrace,g, 865
calledby spadTrace, 866
defun, 865
spadTrace,isTraceable, 865
calledby spadTrace, 866
calls bpiname, 866
calls exit, 866
calls gensymp, 866
calls reportSpadTrace, 866
calls seq, 865
defun, 865
spadTraceAlias, 881
calledby spadTrace, 867
calls internl, 881
defun, 881
spadUntrace, 885
calls assoc, 885
calls bpiname, 886
calls bpiuntrace, 886
calls bright, 886
calls delasc, 886
calls devaluate, 885
calls exit, 886
calls getOption, 885
calls isDomainOrPackage, 885
calls prefix2String, 886
calls remover, 886
calls rplac, 886
calls sayMSG, 885
calls seq, 886
calls setletprintflag, 886
calls spadReply, 886
calls userError, 885
uses /tracenames, 886
uses $letAssoc, 886
defun, 885
specialChar, 970
calledby displayOperationsFromLisplib, 812
calledby displaySetOptionInformation, 647
calledby displaySetVariableSettings, 649
calledby filterAndFormatConstructors, 934

calledby printSynonyms, 470
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calledby setExposeAddConstr, 692
calledby setExposeAddGroup, 691
calledby setExposeAdd, 689
calledby setExposeDropConstr, 695
calledby setExposeDropGroup, 694
calledby setExposeDrop, 693
calledby setOutputCharacters, 758
calledby spadStartUpMsgs, 19
calledby whatCommands, 932
calledby workfilesSpad2Cmd, 939
calls assq, 970
calls ifcdr, 970
uses $specialCharacterAlist, 970
uses $specialCharacters, 970
defun, 970
spleI, 119
calledby scanExponent, 124
calledby scanNumber, 129
calledby scanPossFloat, 118
calls spleI1, 119
defun, 119
spleI1, 120
calledby scanNumber, 129
calledby spleI1, 120
calledby spleI, 119
calls concat, 120
calls qenum, 120
calls scanEsc, 120
calls spleI1, 120
calls substring, 120
uses $ln, 120
uses $n, 120
uses $sz, 120
defun, 120
splitIntoOptionBlocks, 443
calledby doSystemCommand, 442
calls stripSpaces, 443
defun, 443
spool help page, 819
manpage, 819
squeeze, 1017
calledby write-browsedb, 1024
calledby write-categorydb, 1025
calledby write-interpdb, 1022
calledby write-operationdb, 1026
uses *compressvector*, 1017
defun, 1017
ST
calledby intloopInclude, 63
stackTraceOptionError, 851
calledby getTraceOption,hn, 844
calledby getTraceOption, 844
calledby traceOptionError, 847
calledby transOnlyOption, 850
uses $traceErrorStack, 851
defun, 851
startsComment?, 113
calledby scanEsc, 121
calledby scanToken, 111
calls qenum, 113
uses $ln, 113
uses $n, 113
uses $sz, 113
defun, 113
startsId?, 1038
calledby scanToken, 111
defmacro, 1038
startsNegComment?, 114
calledby scanEsc, 121
calledby scanToken, 111
calls qenum, 114
uses $ln, 114
uses $n, 114
uses $sz, 114
defun, 114
startTimingProcess
calledby evalDomain, 903
calledby loadLib, 1029
calledby localnrlib, 1007
calledby processInteractive1, 52
calledby recordNewValue, 587
calledby recordOldValue, 588
calledby updateHist, 585
statisticsInitialization
calledby restart, 17
calls gbc-time, 1029
statisticsSummary
calledby printStatisticsSummary, 57
stopTimingProcess

calledby evalDomain, 903
calledby interpretTopLevel, 53
calledby loadLibNoUpdate, 1031
calledby loadLib, 1030
calledby processInteractive1, 52
calledby recordNewValue, 587
calledby recordOldValue, 588
calledby updateHist, 585
strconc
calledby displayMacro, 450
calledby displayValue, 455
calledby editFile, 541
calledby fixObjectForPrinting, 452
calledby makeMsgFromLine, 369
calledby processChPosesForOneLine, 374
calledby processSynonyms, 33
calledby reportOpsFromLisplib, 810
calledby reportOpsFromUnitDirectly, 814
calledby whatCommands, 932
streamChop, 72
calledby ncloopDQlines, 71
calledby streamChop, 72
calls StreamNull, 72
calls ncloopPrefix?, 72
calls streamChop, 72
defun, 72
StreamNil, 104
usedby incRgen1, 104
defvar, 104
StreamNull, 331
calledby incAppend1, 87
calledby incLude1, 81
calledby incZip1, 74
calledby intloopProcess, 64
calledby ncloopDQlines, 71
calledby next1, 38
calledby npNull, 331
calledby parseFromString, 48
calledby streamChop, 72
calls eqcar, 331
defun, 331
string-concatenate
calledby concat, 1041
string2id-n
calledby close, 507
calledby historySpad2Cmd, 578
calledby importFromFrame, 559
calledby listConstructorAbbreviations, 480
calledby processSynonyms, 33
calledby quitSpad2Cmd, 634
calledby synonymsForUserLevel, 825
calledby yesanswer, 533
string2pint-n
calledby getAliasIfTracedMapParameter, 879
string2Words
calledby segmentKeyedMsg, 328
stringimage
calledby constructSubst, 432
stringMatches?, 1075
calls basicMatch?, 1075
defun, 1075
stringp
calledby porigin, 88
stringPrefix?
calledby clearCmdExcept, 500
calledby hasOption, 447
calledby removeUndoLines, 923
calledby selectOption, 475
calledby undo, 912
stringSpaces
calledby getFirstWord, 465
StringToDir, 1076
calledby fnameMake, 1075
calls lastc, 1076
defun, 1076
stripLisp, 467
calledby handleNoParseCommands, 466
defun, 467
stripSpaces, 467
calledby dumbTokenize, 463
calledby handleNoParseCommands, 466
calledby parseSystemCmd, 465
calledby splitIntoOptionBlocks, 443
defun, 467
strpos, 1039
calledby getSystemCommandLine, 825
calledby processSynonyms, 33
calledby rdigit?, 131
calledby satisfiesRegularExpressions, 933
calledby scanS, 128
calledby stupidIsSpadFunction, 896

defun, 1039
strposl, 1040
calledby nextline, 110
calledby scanEsc, 121
calledby scanSpace, 127
defun, 1040
stupidIsSpadFunction, 896
calledby breaklet, 895
calledby tracelet, 894
calls pname, 896
calls strpos, 896
defun, 896
subCopy
calledby domArg2, 420
calledby hasAtt, 422
calledby hasCate, 431
calledby hasCaty, 418
calledby hasSigAnd, 423
calledby hasSig, 421
calledby replaceSharps, 948
calledby unifyStructVar, 427
sublislis
calledby localnrlib, 1007
subMatch, 116
calledby scanPunct, 115
calls substringMatch, 116
defun, 116
subseq
calledby getFirstWord, 465
substituteSegmentedMsg
calledby getMsgInfoFromKey, 364
calledby msgText, 62
calledby sayKeyedMsgLocal, 328
substring
calledby ExecuteInterpSystemCommand, 33
calledby alqlGetKindString, 1074
calledby alqlGetOrigin, 1073
calledby alqlGetParams, 1074
calledby doSystemCommand, 442
calledby getAliasIfTracedMapParameter, 879
calledby getSystemCommandLine, 825
calledby incDrop, 101
calledby lineoftoks, 109
calledby mkprompt, 42
calledby printLabelledList, 470
calledby processSynonyms, 33
calledby removeUndoLines, 923
calledby scanComment, 113
calledby scanNegComment, 114
calledby scanS, 128
calledby scanW, 125
calledby spleI1, 120
calledby writeInputLines, 583
substringMatch, 116
calledby subMatch, 116
calls qenum, 117
calls size, 117
defun, 116
subTypes, 856
calledby spadTrace, 867
calledby subTypes, 856
calls exit, 856
calls lassoc, 856
calls seq, 856
calls subTypes, 856
defun, 856
suchthat, 404
syntax, 404
summary, 822
calls concat, 822
calls getenviron, 822
calls obey, 822
defun, 822
summary help page, 821
manpage, 821
syGeneralErrorHere, 190
calledby npListAndRecover, 187
calledby npTrapForm, 210
calls sySpecificErrorHere, 190
defun, 190
syIgnoredFromTo, 189
calledby npRecoverTrap, 188
calls FromTo, 189
calls From, 189
calls To, 189
calls ncSoftError, 189
calls pfGlobalLinePosn, 189
defun, 189
synonym, 824
calls synonymSpad2Cmd, 824

defun, 824
synonym help page, 823
manpage, 823
synonymsForUserLevel, 825
calledby printSynonyms, 470
calls commandsForUserLevel, 825
calls selectOptionLC, 825
calls string2id-n, 825
uses $UserLevel, 825
uses $systemCommands, 825
defun, 825
synonymSpad2Cmd, 824
calledby synonym, 824
calls getSystemCommandLine, 824
calls printSynonyms, 824
calls processSynonymLine, 824
calls putalist, 824
calls terminateSystemCommand, 824
uses $CommandSynonymAlist, 824
defun, 824
syntax
assignment, 381
blocks, 384
clef, 386
collection, 387
for, 389
if, 393
iterate, 395
leave, 396
parallel, 397
repeat, 400
suchthat, 404
while, 405
sySpecificErrorAtToken, 190
calledby sySpecificErrorHere, 190
calls ncSoftError, 190
calls tokPosn, 190
defun, 190
sySpecificErrorHere, 190
calledby syGeneralErrorHere, 190
calls sySpecificErrorAtToken, 190
uses $stok, 190
defun, 190
system help page, 829
manpage, 829
systemCommand, 444
calledby handleParsedSystemCommands, 464
calledby handleTokensizeSystemCommands, 443
calls commandsForUserLevel, 444
calls helpSpad2Cmd, 444
calls selectOptionLC, 444
calls selectOption, 444
uses $CategoryFrame, 444
uses $e, 444
uses $options, 444
uses $syscommands, 444
uses $systemCommands, 444
defun, 444
systemError
calledby pfDefinition2Sex, 313
calledby pfLambdaTran, 314
systemErrorHere
calledby interpret2, 55
calledby reportOpsFromUnitDirectly, 814

tab
calledby showInput, 595
tabbing, 360
calledby getStFromMsg, 352
calls getMsgPrefix?, 360
uses $preLength, 360
defun, 360
take
calledby ?t, 893
terminateSystemCommand, 448
calledby commandAmbiguityError, 448
calledby commandErrorMessage, 446
calledby displayProperties, 458
calledby npProcessSynonym, 469
calledby setFortPers, 748
calledby setStreamsCalculate, 793
calledby synonymSpad2Cmd, 824
calledby userLevelErrorMessage, 447
calls tersyscommand, 448
defun, 448
tersyscommand, 448
calledby library, 1005
calledby quitSpad2Cmd, 634
calledby terminateSystemCommand, 448
calls spadThrow, 448

defun, 448
The restart function, 16
thefname, 91
calledby inclmsgCannotRead, 92
calledby inclmsgNoSuchFile, 91
calls pfname, 91
defun, 91
theid, 90
defun, 90
theorigin, 88
defun, 88
thisPosIsEqual, 372
calls poGlobalLinePosn, 372
calls poNopos?, 372
defun, 372
thisPosIsLess, 372
calls poGlobalLinePosn, 372
calls poNopos?, 372
defun, 372
throwEvalTypeMsg, 909
calledby evaluateType1, 907
calledby evaluateType, 906
calls spadThrow, 909
calls throwKeyedMsg, 909
local ref $noEvalTypeMsg, 909
defun, 909
throwKeyedMsg
calledby addNewInterpreterFrame, 557
calledby closeInterpreterFrame, 558
calledby close, 506
calledby fetchOutput, 596
calledby frameSpad2Cmd, 562
calledby getTraceOptions, 842
calledby getTraceOption, 844
calledby importFromFrame, 559
calledby readSpad2Cmd, 638
calledby restoreHistory, 593
calledby saveHistory, 591
calledby spadrwrite, 601
calledby throwEvalTypeMsg, 909
calledby trace1, 838
calledby transTraceItem, 853
calledby workfilesSpad2Cmd, 939
calledby writeInputLines, 583
throwKeyedMsgCannotCoerceWithValue
calledby evaluateType1, 908
calledby interpret2, 55
throwListOfKeyedMsgs
calledby getTraceOptions, 842
throws
cacheKeyedMsg, 327
fin, 544
intloopReadConsole, 30
monitor-file, 1153
monitor-readinterp, 1160
monitor-spadfile, 1162
ncError, 69
npMissing, 149
npTrap, 210
npTrapForm, 210
ScanOrPairVec,ScanOrInner, 611
spad-syntax-error, 959
writify,writifyInner, 603
timedEVALFUN
calledby getAndEvalConstructorArgument, 948
tmp1
calledby diffAlist, 915
To, 378
calledby syIgnoredFromTo, 189
defun, 378
toFile?, 361
calledby msgOutputter, 351
calls getMsgToWhere, 361
uses $fn, 361
defun, 361
tokConstruct, 409
calledby enPile, 338
calledby npDDInfKey, 206
calledby npDollar, 181
calledby npFirstTok, 141
calledby npId, 202
calledby npInfixOperator, 158
calledby npPrefixColon, 159
calledby npPushId, 207
calledby npSymbolVariable, 203
calledby pfLeaf, 245
calledby separatePiles, 339
calls ifcar, 409
calls ncPutQ, 409
calls pfNoPosition?, 409
defun, 409

token-stack-show, 961
calledby iostat, 960
calls token-type, 961
uses current-token, 961
uses next-token, 961
uses prior-token, 961
uses valid-tokens, 961
defun, 961
token-type
calledby token-stack-show, 961
tokPart, 411
calledby intloopProcess, 64
calledby npDDInfKey, 206
calledby npFirstTok, 141
calledby npInfixOperator, 158
calledby npSymbolVariable, 203
calledby pf2Sex1, 300
calledby pfCopyWithPos, 234
calledby pfIdSymbol, 245
calledby pfLeafToken, 246
calledby pfLiteralString, 247
calledby pfSexpr,strip, 248
calledby pfSymbolSymbol, 250
calledby pfSymb, 249
calledby pileCforest, 338
defun, 411
tokPosn, 411
calledby firstTokPosn, 339
calledby lastTokPosn, 339
calledby ncloopDQlines, 71
calledby npConstTok, 182
calledby npDDInfKey, 206
calledby npDollar, 181
calledby npFirstTok, 141
calledby npId, 203
calledby npInfixOperator, 158
calledby npMissingMate, 213
calledby npMissing, 149
calledby npParse, 139
calledby npPrefixColon, 159
calledby npPushId, 207
calledby npRecoverTrap, 188
calledby npSymbolVariable, 203
calledby npTrapForm, 210
calledby npTrap, 210
calledby pfBraceBar, 255
calledby pfBrace, 255
calledby pfBracketBar, 256
calledby pfBracket, 255
calledby pfLeafPosition, 246
calledby pileColumn, 335
calledby sySpecificErrorAtToken, 190
calls ncAlist, 411
calls pfNoPosition, 411
calls qassq, 411
defun, 411
tokTran, 463
calledby handleParsedSystemCommands, 464
calledby handleTokensizeSystemCommands, 443
calledby parseSystemCmd, 465
calls isIntegerString, 463
defun, 463
tokType, 411
calledby npConstTok, 182
calledby pilePlusComment, 334
calls ncTag, 411
defun, 411
Top, 77
usedby incStream, 73
usedby incString, 39
defvar, 77
Top?, 79
calledby incLude1, 81
calledby xlIfSyntax, 96
calls quotient, 79
defvar, 79
toplevel
calledby loadLibNoUpdate, 1031
toScreen?, 349
calledby msgOutputter, 351
calls getMsgToWhere, 349
defun, 349
TPDHERE
Beware that this function occurs with lowercase also, 163
Beware that this function occurs with uppercase also, 162
Make this more international, not EBCDIC, 970

Note that there is also an npADD function, 157
Note that there is also an npAdd function, 156
Remove all boot references from top level, 468
The pform function has a leading percent sign. fix this, 66, 219, 221, 223, 227
The variable al is undefined, 363
This could probably be replaced by the default assoc using eql, 1044
This function should be replaced by fillerspaces, 369
This should probably be a macro or eliminated, 467
Well this makes no sense., 195
getMsgFTTag is nonsense, 376
note that the file interp.exposed no longer exists., 1160
rewrite this using (dolist (item seqList)...), 308
rewrite using dsetq, 309
trace, 837
calledby ltrace, 628
calls traceSpad2Cmd, 837
defun, 837
trace help page, 831
manpage, 831
trace1, 838
calledby trace1, 838
calledby traceSpad2Cmd, 837
calls /trace,0, 838
calls ?t, 838
calls addassoc, 838
calls centerAndHighlight, 838
calls delete, 838
calls devaluate, 838
calls exit, 838
calls getTraceOptions, 838
calls getTraceOption, 838
calls hasOption, 838
calls isFunctor, 838
calls lassoc, 838
calls pcounters, 838
calls poundsign, 838
calls ptimers, 838
calls qcar, 838
calls qcdr, 838
calls qslessp, 838
calls resetCounters, 838
calls resetSpacers, 838
calls resetTimers, 838
calls saveMapSig, 838
calls sayKeyedMsg, 838
calls say, 838
calls selectOptionLC, 838
calls seq, 838
calls throwKeyedMsg, 838
calls trace1, 838
calls transTraceItem, 838
calls unabbrev, 838
calls untraceDomainLocalOps, 838
calls untrace, 838
calls vecp, 838
uses $lastUntraced, 838
uses $optionAlist, 839
uses $options, 838
uses $traceNoisely, 838
defun, 838
traceDomainConstructor, 870
calls concat, 871
calls embed, 871
calls exit, 871
calls getOption, 870
calls loadFunctor, 871
calls mkq, 871
calls seq, 870
calls spadTrace, 871
calls traceDomainLocalOps, 871
uses $ConstructorCache, 871
defun, 870
traceDomainLocalOps, 870
calledby traceDomainConstructor, 871
calls sayMSG, 870
defun, 870
tracelet, 894
calls bpiname, 894
calls compileBoot, 894
calls delete, 894
calls gensymp, 894
calls isgenvar, 894

calls lassoc, 894
calls setletprintflag, 894
calls stupidIsSpadFunction, 894
calls union, 894
uses $QuickLet, 894
uses $letAssoc, 894
uses $traceletFunctions, 894
uses $traceletflag, 894
defun, 894
traceOptionError, 847
calls commandAmbiguityError, 847
calls stackTraceOptionError, 847
defun, 847
traceReply, 889
calledby traceSpad2Cmd, 837
calls abbreviate, 889
calls addTraceItem, 889
calls concat, 889
calls exit, 889
calls flowSegmentedMsg, 889
calls isDomainOrPackage, 889
calls isFunctor, 889
calls isSubForRedundantMapName, 889
calls isgenvar, 889
calls poundsign, 889
calls prefix2String, 889
calls qcar, 889
calls rassocSub, 889
calls sayBrightlyLength, 889
calls sayBrightly, 889
calls sayMSG, 889
calls sayMessage, 889
calls seq, 889
calls userError, 889
uses /tracenames, 890
uses $constructors, 889
uses $domains, 889
uses $linelength, 889
uses $packages, 889
defun, 889
traceSpad2Cmd, 837
calledby trace, 837
calls augmentTraceNames, 837
calls getMapSubNames, 837
calls qcar, 837
calls qcdr, 837
calls trace1, 837
calls traceReply, 837
uses $mapSubNameAlist, 837
defun, 837
trademark, 519
defun, 519
translateTrueFalse2YesNo, 651
calledby displaySetOptionInformation, 647
calledby displaySetVariableSettings, 649
defun, 651
translateYesNo2TrueFalse, 650
calledby initializeSetVariables, 645
calledby set1, 801
calls member, 650
defun, 650
transOnlyOption, 850
calledby getTraceOption, 844
calledby transOnlyOption, 850
calls qcar, 850
calls qcdr, 850
calls stackTraceOptionError, 850
calls transOnlyOption, 850
calls upcase, 850
defun, 850
transTraceItem, 853
calledby trace1, 838
calledby transTraceItem, 853
calledby untrace, 852
calls constructor?, 853
calls devaluate, 853
calls domainToGenvar, 853
calls get, 853
calls member, 853
calls objMode, 853
calls objVal, 853
calls throwKeyedMsg, 853
calls transTraceItem, 853
calls unabbrev, 853
calls vecp, 853
uses $doNotAddEmptyModeIfTrue, 853
defun, 853
trapNumericErrors, 1065
defmacro, 1065
tree
calledby displaySetVariableSettings, 649
calledby initializeSetVariables, 645

calledby set1, 801
trimString
calledby removeUndoLines, 923
truth-to-bit, 1056
defmacro, 1056
typeIsASmallInteger
calledby defaultTargetFE, 435
types
calledby clearCmdParts, 501

unabbrev
calledby reportOperations, 808
calledby setExposeAddConstr, 692
calledby setExposeDropConstr, 695
calledby trace1, 838
calledby transTraceItem, 853
unabbrevAndLoad
calledby domainToGenvar, 851
unAbbreviateKeyword, 465
calledby doSystemCommand, 442
calls commandsForUserLevel, 465
calls selectOptionLC, 465
calls selectOption, 465
uses $currentLine, 465
uses $syscommands, 465
uses $systemCommands, 465
uses line, 466
defun, 465
underbar
usedby writeInputLines, 583
underDomainOf
calledby hasCateSpecialNew, 433
undo, 912
calls identp, 912
calls pname, 912
calls qcar, 912
calls qcdr, 912
calls read, 912
calls spaddifference, 912
calls stringPrefix?, 912
calls undoCount, 912
calls undoSteps, 912
calls userError, 912
uses $InteractiveFrame, 912
uses $options, 912
defun, 912
undo help page, 899
manpage, 899
undoChanges, 589
calledby undoChanges, 589
calledby undoInCore, 588
calls boot-equal, 589
calls putHist, 589
calls undoChanges, 589
uses $HistList, 589
uses $InteractiveFrame, 590
defun, 589
undoCount, 919
calledby removeUndoLines, 924
calledby undo, 912
calls concat, 919
calls spaddifference, 919
calls userError, 919
uses $IOindex, 919
defun, 919
undoFromFile, 590
calls assq, 590
calls disableHist, 590
calls exit, 590
calls putHist, 590
calls readHiFi, 590
calls recordNewValue, 590
calls recordOldValue, 590
calls seq, 590
calls updateHist, 590
uses $HiFiAccess, 590
uses $InteractiveFrame, 590
defun, 590
undoInCore, 588
calls assq, 589
calls disableHist, 589
calls putHist, 589
calls readHiFi, 589
calls sayKeyedMsg, 589
calls spaddifference, 588
calls undoChanges, 588
calls updateHist, 589
uses $HiFiAccess, 589
uses $HistListLen, 589
uses $HistList, 589
uses $IOindex, 589
uses $InteractiveFrame, 589

defun, 588
undoLocalModemapHack, 923
calledby undoSingleStep, 921
calls exit, 923
calls seq, 923
defun, 923
undoSingleStep, 921
calledby undoSteps, 920
calls assq, 921
calls exit, 921
calls lassoc, 921
calls seq, 921
calls undoLocalModemapHack, 921
defun, 921
undoSteps, 920
calledby undo, 912
calls copy, 920
calls qcar, 920
calls qcdr, 920
calls recordFrame, 920
calls spaddifference, 920
calls undoSingleStep, 920
calls writeInputLines, 920
uses $IOindex, 920
uses $InteractiveFrame, 920
uses $frameRecord, 920
defun, 920
unembed
calledby untraceDomainConstructor, 873
unescapeStringsInForm, 62
calledby serverReadLine, 44
calledby unescapeStringsInForm, 62
calls unescapeStringsInForm, 62
uses $funnyBacks, 62
uses $funnyQuote, 62
defun, 62
unifyStruct, 426
calledby hasAtt, 422
calledby hasCaty, 418
calledby hasSig, 421
calledby unifyStructVar, 427
calledby unifyStruct, 426
calls isPatternVar, 426
calls unifyStructVar, 426
calls unifyStruct, 426
defun, 426
unifyStructVar, 427
calledby unifyStruct, 426
calls augmentSub, 427
calls canCoerce, 427
calls constructor?, 427
calls contained, 427
calls containsVars, 427
calls isPatternVar, 427
calls lassoc, 427
calls resolveTT, 427
calls subCopy, 427
calls unifyStruct, 427
local def $hope, 427
local ref $Coerce, 427
local ref $Subst, 427
local ref $domPvar, 427
defun, 427
union
calledby breaklet, 895
calledby getMapSubNames, 859
calledby tracelet, 894
unionq
calledby getMapSubNames, 859
unsqueeze, 1018
calledby browseOpen, 996
calledby categoryOpen, 997
calledby getdatabase, 1001
calledby interpOpen, 995
calledby operationOpen, 998
uses *compressvector*, 1018
defun, 1018
untrace, 852
calledby trace1, 838
calls /untrace,0, 852
calls copy, 852
calls lassocSub, 852
calls removeTracedMapSigs, 852
calls transTraceItem, 852
uses /tracenames, 852
uses $lastUntraced, 852
uses $mapSubNameAlist, 852
defun, 852
untraceDomainConstructor, 873
calls concat, 873
calls delete, 873
calls exit, 873

calls seq, 873
calls unembed, 873
calls untraceDomainConstructor,keepTraced?, 873
uses /tracenames, 873
defun, 873
untraceDomainConstructor,keepTraced?, 872
calledby untraceDomainConstructor, 873
calls /untrace,0, 872
calls boot-equal, 872
calls devaluate, 872
calls exit, 872
calls isDomainOrPackage, 872
calls kar, 872
calls qcar, 872
calls seq, 872
defun, 872
untraceDomainLocalOps, 870
calledby trace1, 838
calls sayMSG, 870
defun, 870
untraceMapSubNames, 863
calledby clearCmdAll, 499
calledby clearCmdParts, 501
calls /untrace,2, 863
calls assocright, 863
calls getPreviousMapSubNames, 863
calls setdifference, 863
uses $lastUntraced, 863
uses $mapSubNameAlist, 863
defun, 863
unwritable?, 602
calls placep, 602
calls vecp, 602
defun, 602
upcase
calledby close, 506
calledby historySpad2Cmd, 578
calledby importFromFrame, 559
calledby listConstructorAbbreviations, 480
calledby pathnameTypeId, 1035
calledby quitSpad2Cmd, 634
calledby readSpad2Cmd, 638
calledby setOutputAlgebra, 754
calledby setOutputFormula, 782
calledby setOutputFortran, 761
calledby setOutputHtml, 772
calledby setOutputMathml, 768
calledby setOutputOpenMath, 777
calledby setOutputTex, 788
calledby transOnlyOption, 850
calledby yesanswer, 533
updateCategoryTable
calledby loadLib, 1029
calledby localnrlib, 1007
updateCurrentInterpreterFrame, 555
calledby addNewInterpreterFrame, 557
calledby changeToNamedInterpreterFrame, 556
calledby clearCmdAll, 499
calledby clearSpad2Cmd, 496
calledby previousInterpreterFrame, 557
calledby updateHist, 585
calls createCurrentInterpreterFrame, 555
calls updateFromCurrentInterpreterFrame, 555
uses $interpreterFrameRing, 555
defun, 555
updateDatabase, 1009
calledby loadLib, 1029
calledby localnrlib, 1007
calls clearAllSlams, 1009
calls clearClams, 1009
calls constructor?, 1009
local ref $forceDatabaseUpdate, 1009
defun, 1009
updateFromCurrentInterpreterFrame, 554
calledby addNewInterpreterFrame, 557
calledby changeToNamedInterpreterFrame, 556
calledby closeInterpreterFrame, 558
calledby initializeInterpreterFrameRing, 551
calledby nextInterpreterFrame, 556
calledby previousInterpreterFrame, 557
calledby updateCurrentInterpreterFrame, 555
calls sayMessage, 554
uses $HiFiAccess, 554
uses $HistListAct, 554
uses $HistListLen, 554
uses $HistList, 554

uses $HistRecord, 554
uses $IOindex, 554
uses $InteractiveFrame, 554
uses $frameMessages, 554
uses $internalHistoryTable, 554
uses $interpreterFrameName, 554
uses $interpreterFrameRing, 554
uses $localExposureData, 554
defun, 554
updateHist, 585
calledby processInteractive, 50
calledby undoFromFile, 590
calledby undoInCore, 589
calls disableHist, 585
calls startTimingProcess, 585
calls stopTimingProcess, 585
calls updateCurrentInterpreterFrame, 585
calls updateInCoreHist, 585
calls writeHiFi, 585
uses $HiFiAccess, 586
uses $HistRecord, 586
uses $IOindex, 585
uses $currentLine, 586
uses $mkTestInputStack, 586
defun, 585
updateInCoreHist, 586
calledby restoreHistory, 593
calledby updateHist, 585
uses $HistListAct, 586
uses $HistListLen, 586
uses $HistList, 586
defun, 586
updateSourceFiles, 542
calledby editSpad2Cmd, 540
calledby workfilesSpad2Cmd, 939
calls insert, 542
calls makeInputFilename, 542
calls member, 542
calls pathnameName, 542
calls pathnameTypeId, 542
calls pathnameType, 542
calls pathname, 542
uses $sourceFiles, 542
defun, 542
use-fast-links
calledby set1, 801
userError
calledby spadTrace, 866
calledby spadUntrace, 885
calledby traceReply, 889
calledby undoCount, 919
calledby undo, 912
userLevelErrorMessage, 446
calledby commandUserLevelError, 446
calledby optionUserLevelError, 446
calls commandAmbiguityError, 446
calls sayKeyedMsg, 447
calls terminateSystemCommand, 447
uses $UserLevel, 447
defun, 446

valid-tokens
usedby token-stack-show, 961
validateOutputDirectory, 716
calledby setFortDir, 717
calledby setFortTmpDir, 715
defun, 716
values
calledby clearCmdParts, 501
vec2list, 1049
defun, 1049
vecp
calledby ScanOrPairVec,ScanOrInner, 611
calledby basicLookupCheckDefaults, 1063
calledby basicLookup, 1061
calledby dewritify,dewritifyInner, 608
calledby spadClosure?, 607
calledby trace1, 838
calledby transTraceItem, 853
calledby unwritable?, 602
calledby writify,writifyInner, 603
version
calledby zsystemdevelopment1, 944
vmread
calledby dewritify,dewritifyInner, 608
voidValue, 1047
defun, 1047

warn
calledby getdatabase, 1001
what, 929
calls whatSpad2Cmd, 929

defun, 929
what help page, 927
manpage, 927
whatCommands, 931
calledby whatSpad2Cmd, 930
calls blankList, 932
calls centerAndHighlight, 931
calls commandsForUserLevel, 932
calls filterListOfStrings, 932
calls sayAsManyPerLineAsPossible, 932
calls sayKeyedMsg, 932
calls sayMessage, 932
calls say, 932
calls specialChar, 932
calls strconc, 932
uses $UserLevel, 932
uses $linelength, 932
uses $systemCommands, 932
defun, 931
whatConstructors, 935
calledby filterAndFormatConstructors, 934
calls boot-equal, 935
calls exit, 935
calls getdatabase, 935
calls msort, 935
calls seq, 935
defun, 935
whatSpad2Cmd, 930
calledby listConstructorAbbreviations, 481
calledby whatSpad2Cmd, 930
calledby what, 929
calls apropos, 930
calls exit, 930
calls filterAndFormatConstructors, 930
calls printSynonyms, 930
calls reportWhatOptions, 930
calls sayKeyedMsg, 930
calls selectOptionLC, 930
calls seq, 930
calls whatCommands, 930
calls whatSpad2Cmd,fixpat, 930
calls whatSpad2Cmd, 930
uses $e, 930
uses $whatOptions, 930
defun, 930
whatSpad2Cmd,fixpat, 929
calledby whatSpad2Cmd, 930
calls downcase, 929
calls qcar, 929
defun, 929
whichCat, 363
calledby setMsgForcedAttrList, 362
calledby setMsgUnforcedAttrList, 364
calls ListMember?, 363
uses $attrCats, 363
defun, 363
while, 405, 1033
defmacro, 1033
syntax, 405
whileWithResult, 1034
defmacro, 1034
with, 937
calls library, 937
defun, 937
with help page, 937
manpage, 937
workfiles, 939
calls workfilesSpad2Cmd, 939
defun, 939
workfiles help page, 939
manpage, 939
workfilesSpad2Cmd, 939
calledby workfiles, 939
calls centerAndHighlight, 939
calls delete, 939
calls makeInputFilename, 939
calls namestring, 939
calls pathname, 939
calls sayBrightly, 940
calls sayKeyedMsg, 939
calls say, 939
calls selectOptionLC, 939
calls sortby, 939
calls specialChar, 939
calls throwKeyedMsg, 939
calls updateSourceFiles, 939
uses $linelength, 940
uses $options, 940
uses $sourceFiles, 940
defun, 939
wrap, 1037
calledby wrap, 1037

calls lotsof, 1037
calls wrap, 1037
defun, 1037
write-browsedb, 1024
calledby make-databases, 1010
calls allConstructors, 1024
calls squeeze, 1024
uses *print-pretty*, 1024
uses *sourcefiles*, 1024
uses $spadroot, 1024
defun, 1024
write-categorydb, 1025
calledby make-databases, 1010
calls genCategoryTable, 1025
calls squeeze, 1025
uses *hasCategory-hash*, 1025
uses *print-pretty*, 1025
defun, 1025
write-compress, 1016
calledby make-databases, 1010
calls allConstructors, 1016
calls allOperations, 1016
uses *attributes*, 1016
uses *compress-stream*, 1016
uses *compressVectorLength*, 1016
defun, 1016
write-interpdb, 1022
calledby make-databases, 1010
calls squeeze, 1022
uses *ancestors-hash*, 1022
uses *print-pretty*, 1022
uses $spadroot, 1022
defun, 1022
write-operationdb, 1026
calledby make-databases, 1010
calls squeeze, 1026
uses *operation-hash*, 1026
defun, 1026
write-warmdata, 1027
calledby make-databases, 1010
uses $topicHash, 1027
defun, 1027
writeablep
calledby myWritable?, 1078
writeHiFi, 598
calledby updateHist, 585
calls histFileName, 598
calls object2Identifier, 598
calls rdefiostream, 598
calls rshut, 598
calls spadrwrite, 598
uses $HistRecord, 598
uses $IOindex, 598
uses $currentLine, 598
uses $internalHistoryTable, 598
uses $useInternalHistoryTable, 598
defun, 598
writeHistModesAndValues, 599
calledby processInteractive, 50
calls get, 599
calls putHist, 599
uses $InteractiveFrame, 599
defun, 599
writeInputLines, 583
calledby historySpad2Cmd, 578
calledby saveHistory, 591
calledby undoSteps, 920
calls concat, 583
calls defiostream, 583
calls histFileErase, 583
calls histInputFileName, 583
calls namestring, 583
calls readHiFi, 583
calls sayKeyedMsg, 583
calls shut, 583
calls size, 583
calls spaddifference, 583
calls substring, 583
calls throwKeyedMsg, 583
uses $HiFiAccess, 583
uses $IOindex, 583
uses underbar, 583
defun, 583
writeLib1
calledby saveDependentsHashTable, 1013
calledby saveUsersHashTable, 1014
writify, 606
calledby safeWritify, 603
calls ScanOrPairVec, 606
calls function, 606
calls writify,writifyInner, 606
uses $seen, 606

uses $writifyComplained, 606
defun, 606
writify,writifyInner, 603
calledby writify,writifyInner, 603
calledby writify, 606
calls boot-equal, 603
calls constructor?, 603
calls devaluate, 603
calls exit, 603
calls hashtable-class, 603
calls hget, 603
calls hkeys, 603
calls hput, 603
calls isDomainOrPackage, 603
calls mkEvalable, 603
calls placep, 603
calls qcar, 603
calls qcdr, 603
calls qrplaca, 603
calls qrplacd, 603
calls qsetvelt, 603
calls qvelt, 603
calls qvmaxindex, 603
calls seq, 603
calls spadClosure?, 603
calls vecp, 603
calls writify,writifyInner, 603
uses $NonNullStream, 603
uses $NullStream, 603
uses $seen, 603
defun, 603
throws, 603
writifyComplain, 602
calls sayKeyedMsg, 602
uses $writifyComplained, 602
defun, 602

xlCannotRead, 91
calledby incLude1, 81
calls inclmsgCannotRead, 91
calls xlMsg, 91
defun, 91
xlCmdBug, 98
calledby incLude1, 82
calls inclmsgCmdBug, 98
calls xlMsg, 98
defun, 98
xlConActive, 93
calledby incLude1, 82
calls inclmsgConActive, 93
calls xlMsg, 93
defun, 93
xlConsole, 94
calledby incLude1, 82
calls inclmsgConsole, 94
calls xlMsg, 94
defun, 94
xlConStill, 94
calledby incLude1, 82
calls inclmsgConStill, 94
calls xlMsg, 94
defun, 94
xlFileCycle, 92
calledby incLude1, 81
calls inclmsgFileCycle, 92
calls xlMsg, 92
defun, 92
xlIfBug, 97
calledby incLude1, 82
calls inclmsgIfBug, 97
calls xlMsg, 97
defun, 97
xlIfSyntax, 96
calledby incLude1, 82
calls Else?, 96
calls Top?, 96
calls inclmsgIfSyntax, 96
calls xlMsg, 96
defun, 96
xlMsg, 86
calledby xlCannotRead, 91
calledby xlCmdBug, 98
calledby xlConActive, 93
calledby xlConStill, 94
calledby xlConsole, 94
calledby xlFileCycle, 92
calledby xlIfBug, 97
calledby xlIfSyntax, 96
calledby xlNoSuchFile, 90
calledby xlPrematureEOF, 86
calledby xlPrematureFin, 95
calledby xlSay, 89

calledby xlSkippingFin, 95
calls incLine, 86
defun, 86
xlNoSuchFile, 90
calledby incLude1, 81
calls inclmsgNoSuchFile, 90
calls xlMsg, 90
defun, 90
xlOK, 86
calledby incLude1, 81
calls lxOK1, 86
defun, 86
xlOK1, 86
calledby incLude1, 81
calls incLine1, 86
defun, 86
xlPrematureEOF, 86
calledby incLude1, 81
calls inclmsgPrematureEOF, 86
calls xlMsg, 86
defun, 86
xlPrematureFin, 95
calledby incLude1, 82
calls inclmsgPrematureFin, 95
calls xlMsg, 95
defun, 95
xlSay, 89
calledby incLude1, 81
calls inclmsgSay, 90
calls xlMsg, 89
defun, 89
xlSkip, 89
calledby incLude1, 81
calls CONCAT, 89
calls incLine, 89
defun, 89
xlSkippingFin, 95
calledby incLude1, 82
calls inclmsgFinSkipped, 95
calls xlMsg, 95
defun, 95
xtokenreader, 958
usedby init-boot/spad-reader, 958
defvar, 958

yesanswer, 533
calledby displayOperations, 533
calls queryUserKeyedMsg, 533
calls string2id-n, 533
calls upcase, 533
defun, 533

zeroOneTran, 68
calledby intInterpretPform, 67
calls nsubst, 68
defun, 68
zsystemdevelopment, 943
calls zsystemDevelopmentSpad2Cmd, 943
defun, 943
zsystemdevelopment help page, 943
manpage, 943
zsystemdevelopment1, 944
calledby zsystemDevelopmentSpad2Cmd, 943
calls /D,1, 944
calls /comp, 944
calls bright, 944
calls defiostream, 944
calls kaddr, 944
calls kadr, 944
calls kar, 944
calls next, 944
calls sayBrightly, 944
calls sayMessage, 944
calls selectOptionLC, 944
calls shut, 944
calls version, 944
uses /version, 944
uses /wsname, 944
uses $InteractiveMode, 944
uses $options, 944
catches, 944
defun, 944
zsystemDevelopmentSpad2Cmd, 943
calledby zsystemdevelopment, 943
calls zsystemdevelopment1, 943
uses $InteractiveMode, 943
defun, 943