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 7: Axiom Hyperdoc

Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.

Cyril Alberga
George Andrews
Yurij Baransky
Gilbert Baumslag
Mark Botch
Martin Brock
William Burge
Bruce Char
Gregory V. Chudnovsky
Don Coppersmith
Gary Cornell
Timothy Daly Sr.
Jean Della Dora
Claire DiCrescendo
Martin Dunstan
Robert Edwards
Richard Fateman
Brian Ford
Constantine Frangos
Marc Gaetano
Patricia Gianni
Laureano Gonzalez-Vega
Matt Grayson
Oswald Gschnitzer
Vilya Harvey
Ralf Hemmecke
Pietro Iglio
Grant Keady
Klaus Kusche
Frederic Lehobey
Rudiger Loos
Camm Maguire
Ian Meikle
Gerard Milmeister
Michael Monagan
Mark Murray
John Nelder
Jinzhong Niu
Julian A. Padget
Susan Pelzel
Claude Quitte
Renaud Rioboo
Simon Robinson
Philip Santas
Gerhard Schneider
Fritz Schwarz
Elena Smirnova
Robert Sutor
James Thatcher
Dylan Thurston
Gregory Vanuxem
Juergen Weiss
James Wen
John M. Wiley
Stephen Wilson
Sandra Wityak
Clifford Yapp
Evelyn Zoernack

Roy Adler
Henry Baker
David R. Barton
Fred Blair
Alexandre Bouyer
Manuel Bronstein
Quentin Carpent
Cheekai Chin
Josh Cohen
George Corliss
Meino Cramer
Timothy Daly Jr.
Gabriel Dos Reis
Sam Dooley
Brian Dupee
Heow Eide-Goodman
Bertfried Fauser
Albrecht Fortenbacher
Timothy Freeman
Rudiger Gebauer
Holger Gollan
Stephen Gortler
James Griesmer
Jocelyn Guidry
Satoshi Hamaguchi
Henderson
Richard Jenks
Tony Kennedy
Bernhard Kutzler
Michel Levaud
Michael Lucks
Bob McElrath
David Mentre
Mohammed Mobarak
Marc Moreno-Maza
William Naylor
Godfrey Nolan
Michael O'Connor
Bill Page
Michel Petitot
Norman Ramsey
Jean Rivlin
Michael Rothstein
Alfred Scheerhorn
Martin Schoenert
Nick Simicich
Jonathan Steinbach
Moss E. Sweedler
Baldir Thomas
Barry Trager
Bernhard Wall
M. Weller
Thorsten Werther
Berhard Will
Shmuel Winograd
Waldemar Wiwianka
David Yun
Bruno Zuercher

Richard Anderson
Stephen Balzac
Gerald Baumgartner
Vladimir Bondarenko
Peter A. Broadbery
Florian Bundschuh
Bob Caviness
David V. Chudnovsky
Christophe Conil
Robert Corless
Claire Di Crescenzo
James H. Davenport
Michael Dewar
Lionel Ducos
Dominique Duval
Lars Erickson
Stuart Feldman
George Frances
Korrinn Fu
Kathy Gerber
Teresa Gomez-Diaz
Johannes Grabmeier
Vladimir Grinberg
Steve Hague
Martin Hassner
Antoine Hersen
Kai Kaminski
Paul Kosinski
Larry Lambe
Howard Levy
Richard Luczak
Michael McGettrick
Victor S. Miller
H. Michael Moeller
Scott Morrison
C. Andrew Neff
Arthur Norman
Kostas Oikonomou
Jaap Weel
Didier Pinchon
Michael Richardson
Nicolas Robidoux
Martin Rubey
William Schelter
Marshall Schor
William Sit
Christine Sundaresan
Eugene Surowitz
Mike Thomas
Themos T. Tsikas
Stephen Watt
Mark Wegman
Michael Wester
Clifton J. Williamson
Robert Wisbauer
Knut Wolf
Richard Zippel
Dan Zwillinger

Contents

1 Overview **1**
- 1.1 The Original Plan . . . 2
- 1.2 External Variables . . . 3
- 1.3 hypertex . . . 4
- 1.4 htsearch . . . 4
- 1.5 spadbuf . . . 4
- 1.6 hthits . . . 4
- 1.7 ex2ht . . . 4
- 1.8 htadd . . . 4

2 The hypertex language **5**

3 Shared Code **31**
- 3.0.1 BeStruct . . . 31
- 3.1 Shared Code for file handling . . . 31
  - 3.1.1 strpostfix . . . 31
  - 3.1.2 extendHT . . . 32
  - 3.1.3 buildHtFilename . . . 33
  - 3.1.4 pathname . . . 35
  - 3.1.5 htFileOpen . . . 35
  - 3.1.6 dbFileOpen . . . 36
  - 3.1.7 tempFileOpen . . . 37
- 3.2 Shared Code for Hash Table Handling . . . 38
  - 3.2.1 halloc . . . 38
  - 3.2.2 hashInit . . . 39
  - 3.2.3 freeHash . . . 39
  - 3.2.4 hashInsert . . . 40
  - 3.2.5 hashFind . . . 40
  - 3.2.6 hashReplace . . . 41
  - 3.2.7 hashDelete . . . 41
  - 3.2.8 hashMap . . . 41
  - 3.2.9 hashCopyEntry . . . 42
  - 3.2.10 hashCopyTable . . . 42
  - 3.2.11 stringHash . . . 43

3.2.12 stringEqual . . . 43
3.2.13 allocString . . . 43
3.3 Shared Code for Error Handling . . . 44
3.3.1 jump . . . 44
3.3.2 dumpToken . . . 44
3.3.3 printPageAndFilename . . . 45
3.3.4 printNextTenTokens . . . 46
3.3.5 printToken . . . 46
3.3.6 tokenName . . . 47
3.3.7 htperror . . . 49
3.4 Shared Code for Lexical Analyzer . . . 50
3.4.1 parserInit . . . 50
3.4.2 initScanner . . . 51
3.4.3 saveScannerState . . . 51
3.4.4 restoreScannerState . . . 52
3.4.5 ungetChar . . . 52
3.4.6 getChar . . . 53
3.4.7 getChar1 . . . 55
3.4.8 ungetToken . . . 56
3.4.9 getToken . . . 57
3.4.10 pushBeStack . . . 60
3.4.11 checkAndPopBeStack . . . 61
3.4.12 clearBeStack . . . 61
3.4.13 beType . . . 62
3.4.14 beginType . . . 64
3.4.15 endType . . . 65
3.4.16 keywordType . . . 66
3.4.17 getExpectedToken . . . 67
3.4.18 spadErrorHandler . . . 67
3.4.19 resetConnection . . . 68
3.4.20 spadBusy . . . 68
3.4.21 connectSpad . . . 69
3.5 htadd shared code . . . 70
3.6 hypertex shared code . . . 75

4 Shared include files 81
4.1 debug.c . . . 81
4.2 hyper.h . . . 82

5 The spadbuf function 95
5.1 Constants and Headers . . . 95
5.1.1 System includes . . . 95
5.1.2 Local includes . . . 96
5.2 externs . . . 96
5.3 local variables . . . 97
5.4 Code . . . 97

5.4.1 spadbufInterHandler . . . 97
5.4.2 spadbufFunctionChars . . . 98
5.4.3 interpIO . . . 99
5.4.4 . . . 100
5.4.5 main . . . 101

6 The ex2ht function 103
6.1 Constants and Headers . . . 103
6.1.1 System includes . . . 103
6.1.2 Local includes . . . 103
6.2 defines . . . 104
6.3 local variables . . . 104
6.4 Code . . . 104
6.4.1 allocString . . . 104
6.4.2 strPrefix . . . 104
6.4.3 getExTitle . . . 105
6.4.4 exToHt . . . 106
6.4.5 emitHeader . . . 107
6.4.6 emitFooter . . . 107
6.4.7 emitMenuEntry . . . 108
6.4.8 emitSpadCommand . . . 108
6.4.9 openCoverPage . . . 109
6.4.10 closeCoverPage . . . 109
6.4.11 closeCoverFile . . . 109
6.4.12 emitCoverLink . . . 109
6.4.13 addFile . . . 110
6.4.14 main . . . 110

7 The htadd command 111
7.1 Constants and Headers . . . 111
7.1.1 System includes . . . 111
7.1.2 structs . . . 112
7.1.3 Local includes . . . 112
7.1.4 extern references . . . 112
7.1.5 defines . . . 113
7.1.6 forward declarations . . . 113
7.1.7 local variables . . . 114
7.2 The Shared Code . . . 114
7.3 Code . . . 115
7.3.1 parseArgs . . . 115
7.3.2 writable . . . 116
7.3.3 buildDBFilename . . . 117
7.3.4 addfile . . . 119
7.3.5 updateDB . . . 121
7.3.6 addNewPages . . . 123
7.3.7 copyFile . . . 124

7.3.8 getFilename . . . 125
7.3.9 deleteFile . . . 126
7.3.10 deleteDB . . . 127
7.3.11 main . . . 128

8 The hthits function 129
8.1 Constants and Headers . . . 129
8.1.1 System includes . . . 129
8.1.2 defines . . . 130
8.1.3 structs . . . 130
8.1.4 Local includes . . . 130
8.1.5 local variables . . . 130
8.1.6 cmdline . . . 131
8.1.7 handleHtdb . . . 131
8.1.8 handleFile . . . 132
8.1.9 handleFilePages . . . 134
8.1.10 handlePage . . . 135
8.1.11 searchPage . . . 136
8.1.12 squirt . . . 136
8.1.13 splitpage . . . 137
8.1.14 untexbuf . . . 138
8.1.15 badDB . . . 138
8.1.16 regerr . . . 139
8.1.17 main . . . 139

9 The hypertex command 141
9.1 Constants and Headers . . . 141
9.1.1 System includes . . . 141
9.2 structs . . . 142
9.2.1 Local includes . . . 142
9.3 structs . . . 143
9.4 defines . . . 144
9.5 externs . . . 149
9.6 local variables . . . 152
9.7 The Shared Code . . . 157
9.8 Code . . . 162
9.8.1 sigusr2Handler . . . 162
9.8.2 sigcldHandler . . . 162
9.8.3 cleanSocket . . . 162
9.8.4 initHash . . . 163
9.8.5 initPageStructs . . . 163
9.8.6 checkArguments . . . 164
9.8.7 makeServerConnections . . . 166
9.9 Condition Handling . . . 168
9.9.1 insertCond . . . 168
9.9.2 changeCond . . . 168

9.9.3 checkMemostack . . . 169
9.9.4 checkCondition . . . 170
9.10 Dialog Handling . . . 171
9.10.1 redrawWin . . . 171
9.10.2 mystrncpy . . . 171
9.10.3 incLineNumbers . . . 171
9.10.4 decLineNumbers . . . 172
9.10.5 decreaseLineNumbers . . . 172
9.10.6 overwriteBuffer . . . 173
9.10.7 moveSymForward . . . 175
9.10.8 clearCursorline . . . 176
9.10.9 insertBuffer . . . 177
9.10.10 addBufferToSym . . . 179
9.10.11 drawInputsymbol . . . 179
9.10.12 updateInputsymbol . . . 180
9.10.13 drawCursor . . . 181
9.10.14 moveCursorHome . . . 182
9.10.15 moveCursorEnd . . . 182
9.10.16 void moveCursorForward . . . 183
9.10.17 moveCursorDown . . . 184
9.10.18 moveCursorUp . . . 185
9.10.19 clearCursor . . . 186
9.10.20 moveCursorBackward . . . 187
9.10.21 moveRestBack . . . 188
9.10.22 deleteRestOfLine . . . 189
9.10.23 backOverEoln . . . 191
9.10.24 moveBackOneChar . . . 193
9.10.25 backOverChar . . . 195
9.10.26 deleteEoln . . . 196
9.10.27 deleteOneChar . . . 198
9.10.28 deleteChar . . . 199
9.10.29 toughEnter . . . 200
9.10.30 enterNewLine . . . 202
9.10.31 dialog . . . 204
9.11 Format and Display a page . . . 206
9.11.1 showPage . . . 207
9.11.2 exposePage . . . 209
9.11.3 scrollPage . . . 210
9.11.4 pastePage . . . 211
9.12 Event Handling . . . 212
9.12.1 mainEventLoop . . . 213
9.12.2 handleEvent . . . 215
9.12.3 createWindow . . . 217
9.12.4 quitHyperDoc . . . 218
9.12.5 findPage . . . 219
9.12.6 downlink . . . 219

9.12.7 memolink . . . 220
9.12.8 killAxiomPage . . . 220
9.12.9 killPage . . . 220
9.12.10 returnlink . . . 221
9.12.11 uplink . . . 221
9.12.12 windowlinkHandler . . . 222
9.12.13 makeWindowLink . . . 222
9.12.14 lispwindowlinkHandler . . . 222
9.12.15 pasteButton . . . 223
9.12.16 helpForHyperDoc . . . 224
9.12.17 findButtonInList . . . 224
9.12.18 getHyperLink . . . 225
9.12.19 handleButton . . . 226
9.12.20 exitHyperDoc . . . 230
9.12.21 setWindow . . . 231
9.12.22 clearExposures . . . 232
9.12.23 getNewWindow . . . 233
9.12.24 setCursor . . . 235
9.12.25 changeCursor . . . 236
9.12.26 handleMotionEvent . . . 236
9.12.27 initCursorState . . . 236
9.12.28 initCursorStates . . . 237
9.12.29 makeBusyCursor . . . 237
9.12.30 makeBusyCursors . . . 237
9.12.31 HyperDocErrorHandler . . . 237
9.12.32 setErrorHandlers . . . 238
9.13 Line Extent Computation . . . 238
9.13.1 computeInputExtent . . . 238
9.13.2 computePunctuationExtent . . . 239
9.13.3 computeWordExtent . . . 241
9.13.4 computeVerbatimExtent . . . 242
9.13.5 computeSpadsrctxtExtent . . . 242
9.13.6 computeDashExtent . . . 243
9.13.7 computeTextExtent . . . 245
9.13.8 computeBeginItemsExtent . . . 252
9.13.9 computeItemExtent . . . 253
9.13.10 computeMitemExtent . . . 253
9.13.11 endifExtent . . . 253
9.13.12 computeIfcondExtent . . . 254
9.13.13 computeCenterExtent . . . 255
9.13.14 computeBfExtent . . . 256
9.13.15 computeEmExtent . . . 256
9.13.16 computeItExtent . . . 256
9.13.17 computeRmExtent . . . 257
9.13.18 computeButtonExtent . . . 257
9.13.19 endbuttonExtent . . . 258

9.13.20 computePastebuttonExtent 259
9.13.21 endpastebuttonExtent 260
9.13.22 computePasteExtent 260
9.13.23 computeSpadcommandExtent 261
9.13.24 computeSpadsrcExtent 262
9.13.25 endSpadcommandExtent 263
9.13.26 endSpadsrcExtent 264
9.13.27 computeMboxExtent 264
9.13.28 computeBoxExtent 265
9.13.29 computeIrExtent 266
9.13.30 computeImageExtent 267
9.13.31 computeTableExtent 268
9.13.32 computeTitleExtent 269
9.13.33 computeHeaderExtent 270
9.13.34 computeFooterExtent 271
9.13.35 computeScrollingExtent 272
9.13.36 startNewline 273
9.13.37 centerNodes 273
9.13.38 punctuationWidth 274
9.13.39 inputStringWidth 274
9.13.40 wordWidth 274
9.13.41 verbatimWidth 275
9.13.42 widthOfDash 275
9.13.43 textWidth 276
9.13.44 totalWidth 281
9.13.45 initExtents 283
9.13.46 initTitleExtents 283
9.13.47 initText 284
9.13.48 textHeight 284
9.13.49 textHeight1 285
9.13.50 maxX 288
9.13.51 Xvalue 290
9.13.52 trailingSpace 291
9.13.53 insertBitmapFile 292
9.13.54 insertPixmapFile 293
9.13.55 plh 294
9.14 Handling forms 294
9.14.1 computeFormPage 294
9.14.2 windowWidth 295
9.14.3 windowHeight 295
9.14.4 formHeaderExtent 295
9.14.5 formFooterExtent 296
9.14.6 formScrollingExtent 296
9.15 Managing the HyperDoc group stack 297
9.15.1 popGroupStack 297
9.15.2 pushGroupStack 297

9.15.3 initGroupStack . . . 298
9.15.4 emTopGroup . . . 298
9.15.5 rmTopGroup . . . 298
9.15.6 lineTopGroup . . . 299
9.15.7 bfTopGroup . . . 299
9.15.8 ttTopGroup . . . 299
9.15.9 pushActiveGroup . . . 300
9.15.10 pushSpadGroup . . . 300
9.15.11 initTopGroup . . . 300
9.15.12 centerTopGroup . . . 300
9.15.13 copyGroupStack . . . 301
9.15.14 freeGroupStack . . . 301
9.16 Handle input, output, and Axiom communication . . . 302
9.16.1 makeRecord . . . 302
9.16.2 verifyRecord . . . 303
9.16.3 ht2Input . . . 303
9.16.4 makeInputFileName . . . 304
9.16.5 makePasteFileName . . . 304
9.16.6 makeTheInputFile . . . 305
9.16.7 makeInputFileFromPage . . . 306
9.16.8 strCopy . . . 308
9.16.9 inListAndNewer . . . 309
9.16.10 makeInputFileList . . . 310
9.16.11 printPasteLine . . . 311
9.16.12 getSpadOutput . . . 312
9.16.13 getGraphOutput . . . 312
9.16.14 sendCommand . . . 313
9.16.15 printPaste . . . 314
9.16.16 printGraphPaste . . . 315
9.17 X Window window initialization code . . . 316
9.17.1 initializeWindowSystem . . . 316
9.17.2 initTopWindow . . . 318
9.17.3 openFormWindow . . . 320
9.17.4 initFormWindow . . . 322
9.17.5 setNameAndIcon . . . 323
9.17.6 getBorderProperties . . . 324
9.17.7 openWindow . . . 325
9.17.8 setSizeHints . . . 326
9.17.9 getGCs . . . 328
9.17.10 loadFont . . . 329
9.17.11 ingItColorsAndFonts . . . 330
9.17.12 changeText . . . 334
9.17.13 getColor . . . 335
9.17.14 mergeDatabases . . . 337
9.17.15 isIt850 . . . 338
9.18 Handling user page interaction . . . 338

9.18.1 fillBox . . . 338
9.18.2 toggleInputBox . . . 339
9.18.3 toggleRadioBox . . . 339
9.18.4 clearRbs . . . 340
9.18.5 changeInputFocus . . . 340
9.18.6 nextInputFocus . . . 341
9.18.7 prevInputFocus . . . 342
9.18.8 returnItem . . . 343
9.18.9 deleteItem . . . 343
9.19 Manipulate the item stack . . . 344
9.19.1 pushItemStack . . . 344
9.19.2 clearItemStack . . . 344
9.19.3 popItemStack . . . 345
9.19.4 copyItemStack . . . 345
9.19.5 freeItemStack . . . 346
9.20 Keyboard handling . . . 347
9.20.1 handleKey . . . 347
9.20.2 getModifierMask . . . 350
9.20.3 initKeyin . . . 351
9.21 Handle page macros . . . 352
9.21.1 scanHyperDoc . . . 352
9.21.2 number . . . 353
9.21.3 loadMacro . . . 354
9.21.4 initParameterElem . . . 355
9.21.5 pushParameters . . . 356
9.21.6 popParameters . . . 356
9.21.7 parseMacro . . . 357
9.21.8 getParameterStrings . . . 358
9.21.9 parseParameters . . . 360
9.22 Memory management routines . . . 360
9.22.1 freeIfNonNULL . . . 360
9.22.2 allocHdWindow . . . 361
9.22.3 freeHdWindow . . . 362
9.22.4 allocNode . . . 363
9.22.5 freeNode . . . 364
9.22.6 allocIfnode . . . 367
9.22.7 allocCondnode . . . 368
9.22.8 freeCond . . . 368
9.22.9 allocPage . . . 369
9.22.10 freePage . . . 370
9.22.11 freePaste . . . 371
9.22.12 freePastebutton . . . 372
9.22.13 freePastearea . . . 373
9.22.14 freeString . . . 373
9.22.15 freeDepend . . . 373
9.22.16 dontFree . . . 373

9.22.17 freeLines . . . 374
9.22.18 freeInputItem . . . 374
9.22.19 freeInputList . . . 374
9.22.20 freeInputBox . . . 375
9.22.21 freeRadioBoxes . . . 375
9.22.22 allocInputline . . . 375
9.22.23 allocPasteNode . . . 376
9.22.24 allocPatchstore . . . 376
9.22.25 freePatch . . . 376
9.22.26 allocInputbox . . . 377
9.22.27 allocRbs . . . 377
9.22.28 allocButtonList . . . 377
9.22.29 freeButtonList . . . 378
9.22.30 resizeBuffer . . . 378
9.23 Page parsing routines . . . 379
9.23.1 PushMR . . . 379
9.23.2 PopMR . . . 379
9.23.3 loadPage . . . 380
9.23.4 displayPage . . . 381
9.23.5 formatPage . . . 382
9.23.6 parseFromString . . . 382
9.23.7 parseTitle . . . 383
9.23.8 parseHeader . . . 384
9.23.9 initParsePage . . . 385
9.23.10 initParsePatch . . . 385
9.23.11 parsePage . . . 386
9.23.12 parseHyperDoc . . . 387
9.23.13 parsePageFromSocket . . . 395
9.23.14 parsePageFromUnixfd . . . 396
9.23.15 startScrolling . . . 397
9.23.16 startFooter . . . 398
9.23.17 endAPage . . . 399
9.23.18 parseReplacepage . . . 399
9.23.19 windowEqual . . . 400
9.23.20 windowCode . . . 400
9.23.21 windowId . . . 400
9.23.22 readHtDb . . . 401
9.23.23 readHtFile . . . 403
9.23.24 makeLinkWindow . . . 407
9.23.25 makePasteWindow . . . 408
9.23.26 makeSpecialPage . . . 409
9.23.27 main . . . 409
9.23.28 addDependencies . . . 410
9.23.29 isNumber . . . 411
9.23.30 parserError . . . 411
9.23.31 getFilename . . . 412

9.23.32 getInputString . . . 413
9.23.33 getWhere . . . 414
9.23.34 findFp . . . 415
9.24 Handle InputString, SimpleBox, RadioBox input . . . 415
9.24.1 makeInputWindow . . . 416
9.24.2 makeBoxWindow . . . 417
9.24.3 initializeDefault . . . 418
9.24.4 parseInputstring . . . 419
9.24.5 parseSimplebox . . . 421
9.24.6 parseRadiobox . . . 423
9.24.7 addBoxToRbList . . . 425
9.24.8 checkOthers . . . 425
9.24.9 insertItem . . . 426
9.24.10 initPasteItem . . . 426
9.24.11 repasteItem . . . 427
9.24.12 currentItem . . . 427
9.24.13 alreadyThere . . . 428
9.24.14 parseRadioboxes . . . 429
9.25 Routines for paste-in areas . . . 431
9.25.1 parsePaste . . . 431
9.25.2 parsePastebutton . . . 433
9.25.3 parsePatch . . . 435
9.25.4 loadPatch . . . 438
9.26 parsing routines for node types . . . 439
9.26.1 parseIfcond . . . 439
9.26.2 parseCondnode . . . 441
9.26.3 parseHasreturnto . . . 442
9.26.4 parseNewcond . . . 442
9.26.5 parseSetcond . . . 443
9.26.6 parseBeginItems . . . 444
9.26.7 parseItem . . . 445
9.26.8 parseMitem . . . 446
9.26.9 parseVerbatim . . . 447
9.26.10 parseInputPix . . . 448
9.26.11 parseCenterline . . . 449
9.26.12 parseCommand . . . 450
9.26.13 parseButton . . . 451
9.26.14 parseSpadcommand . . . 452
9.26.15 parseSpadsrc . . . 453
9.26.16 parseEnv . . . 454
9.26.17 parseValue1 . . . 455
9.26.18 parseValue2 . . . 456
9.26.19 parseTable . . . 457
9.26.20 parseBox . . . 458
9.26.21 parseMbox . . . 458
9.26.22 parseFree . . . 459

9.26.23 parseHelp . . . 459
9.27 Reading bitmaps . . . 460
9.27.1 HTReadBitmapFile . . . 460
9.27.2 readHot . . . 462
9.27.3 readWandH . . . 463
9.27.4 insertImageStruct . . . 464
9.28 Scrollbar handling routines . . . 465
9.28.1 makeScrollBarWindows . . . 466
9.28.2 drawScroller3DEffects . . . 468
9.28.3 showScrollBars . . . 469
9.28.4 moveScroller . . . 470
9.28.5 drawScrollLines . . . 471
9.28.6 calculateScrollBarMeasures . . . 472
9.28.7 linkScrollBars . . . 473
9.28.8 scrollUp . . . 474
9.28.9 scrollUpPage . . . 474
9.28.10 scrollToFirstPage . . . 475
9.28.11 scrollDown . . . 475
9.28.12 scrollDownPage . . . 476
9.28.13 scrollScroller . . . 477
9.28.14 hideScrollBars . . . 478
9.28.15 getScrollBarMinimumSize . . . 478
9.28.16 ch . . . 478
9.28.17 changeWindowBackgroundPixmap . . . 478
9.29 Display text object . . . 479
9.29.1 showText . . . 480
9.29.2 showLink . . . 486
9.29.3 showPaste . . . 487
9.29.4 showPastebutton . . . 488
9.29.5 showInput . . . 488
9.29.6 showSimpleBox . . . 489
9.29.7 showSpadcommand . . . 490
9.29.8 showImage . . . 491
9.30 Axiom communication interface . . . 493
9.30.1 issueSpadcommand . . . 493
9.30.2 sendPile . . . 494
9.30.3 issueDependentCommands . . . 495
9.30.4 markAsExecuted . . . 496
9.30.5 startUserBuffer . . . 497
9.30.6 clearExecutionMarks . . . 498
9.30.7 acceptMenuConnection . . . 499
9.30.8 acceptMenuServerConnection . . . 500
9.30.9 printToString . . . 501
9.30.10 printToString1 . . . 502
9.30.11 issueServerCommand . . . 508
9.30.12 issueServerpaste . . . 509

9.30.13 issueUnixcommand 509
9.30.14 issueUnixlink 510
9.30.15 issueUnixpaste 510
9.30.16 serviceSessionSocket 511
9.30.17 switchFrames 511
9.30.18 sendLispCommand 512
9.30.19 escapeString 512
9.30.20 unescapeString 512
9.30.21 closeClient 513
9.30.22 printSourceToString 514
9.30.23 printSourceToString1 515
9.31 Produce titlebar 524
9.31.1 makeTitleBarWindows 524
9.31.2 showTitleBar 525
9.31.3 linkTitleBarWindows 527
9.31.4 readTitleBarImages 528
9.31.5 getTitleBarMinimumSize 529
9.31.6 main 530

10 The htsearch script 533

11 The presea script 535
11.1 token.h 537

12 The Bitmaps 543
12.1 ht_icon 543
12.2 exit.bitmap 544
12.3 help2.bitmap 545
12.4 return3.bitmap 546
12.5 up3.bitmap 547
12.6 noop.bitmap 548
12.7 exit3d.bitmap 549
12.8 help3d.bitmap 550
12.9 home3d.bitmap 551
12.10 up3d.bitmap 552
12.11 noop3d.bitmap 553

13 Makefile 555

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))

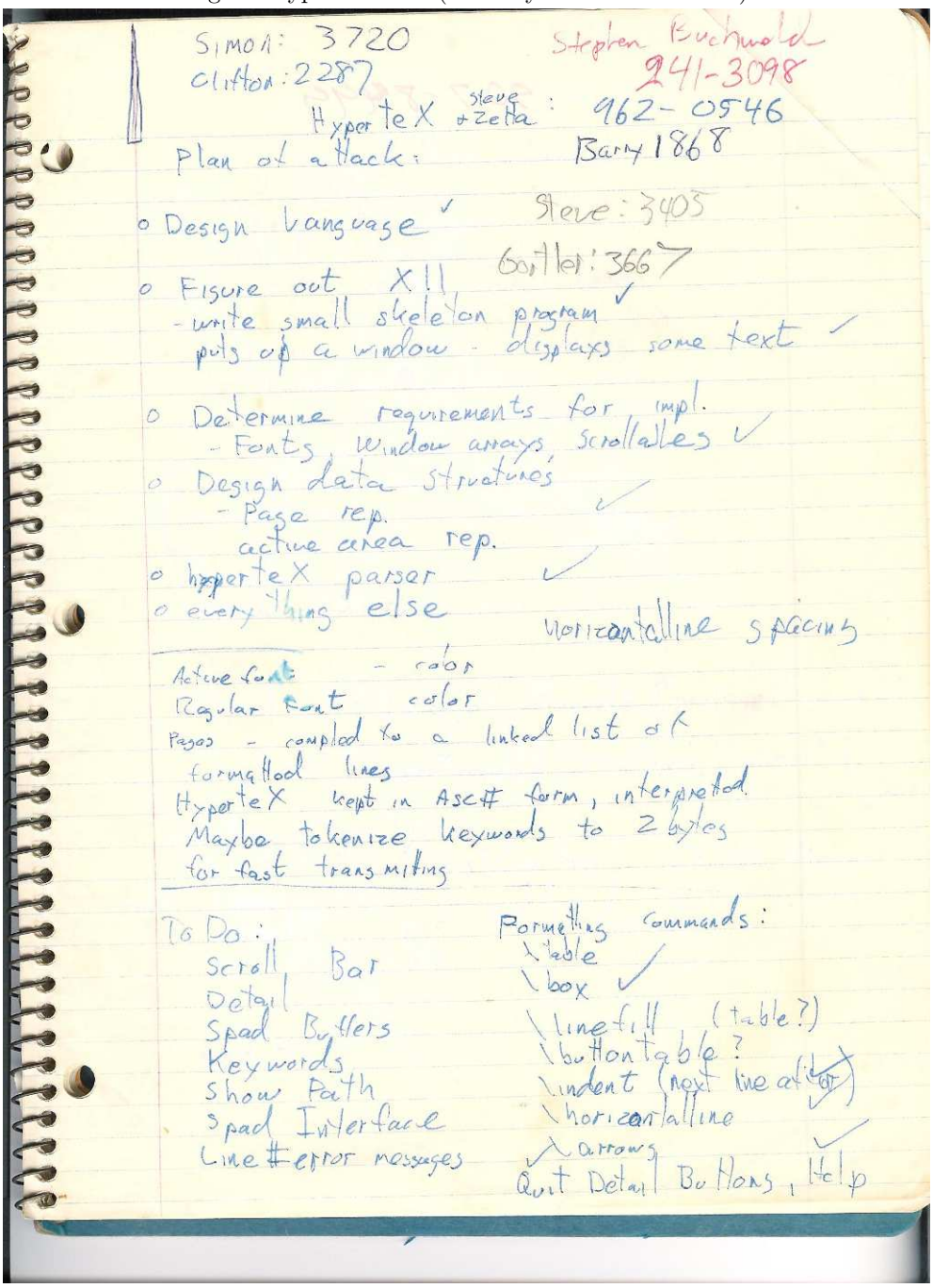
Chapter 1

Overview

This book covers 5 top level commands that make up the Axiom Hyperdoc browser. The primary command is the hypertex command which can be run as a standalone program to browse the Axiom documentation. It can also be run by Axiom to enable lookup of information in the Axiom runtime.

1.1 The Original Plan

The Original Hypertex Plan (courtesy of Scott Morrison)



Simon: 3720
Clifton: 2287
Stephen Buchwald 241-3098
HyperteX steve + zeta: 962-0546
Barry 1868
Plan of attack:
Steve: 3405
Gaitler: 3667

- Design language ✓
- Figure out X11 ✓
  - write small skeleton program puts up a window - displays some text ✓
- Determine requirements for impl.
  - Fonts, window arrays, scrollables ✓
- Design data structures ✓
  - Page rep.
  - active area rep.
- hyperteX parser ✓
- every thing else

horizontalline spacing

Active font - color
Regular Font color
Pages - compiled to a linked list of formatted lines
HyperteX kept in Ascii form, interpreted
Maybe tokenize keywords to 2 bytes for fast transmitting

To Do:
- Scroll Bar
- Detail
- Spad Buffers
- Keywords
- Show Path
- Spad Interface
- Line # error messages

Formatting commands:
- \table
- \box ✓
- \linefill (table?)
- \buttontable?
- \indent (next line after) ✓
- \horizontalline ✓
- ✓ \arrows
- Quit Detail Buttons, Help

1.2 External Variables

Not mentioned elsewhere,

- the HTPATH shell variable, if set, is used to resolve page path names.
- the HTASCII shell variable, if set, is used to choose between ascii and the IBM Code Page 850 character set. See initScanner 3.4.2 on page 51
- the XENVIRONMENT shell variable, if set is used to find the X database to merge, otherwise it uses .Xdefaults from the HOME directory. See 9.17.14 on page 337
- NOFREE shell variable is supposed to turn off freeing memory. See 9.22.10 on page 370
- SPADNUM shell variable is the number of the spad communication socket. See 9.30.2 on page 494

The Axiom user properties in `$HOME/.Xdefaults.` can contain these initialization names:

- Axiom.hyperdoc.FormGeometry
- Axiom.hyperdoc.Geometry
- Axiom.hyperdoc.ActiveColor
- Axiom.hyperdoc.Background
- Axiom.hyperdoc.EmphasizeColor
- Axiom.hyperdoc.EmphasizeFont
- Axiom.hyperdoc.Foreground
- Axiom.hyperdoc.InputBackground
- Axiom.hyperdoc.InputForeground
- Axiom.hyperdoc.SpadColor
- Axiom.hyperdoc.SpadFont
- Axiom.hyperdoc.RmFont
- Axiom.hyperdoc.TtFont
- Axiom.hyperdoc.ActiveFont
- Axiom.hyperdoc.AxiomFont

- Axiom.hyperdoc.SpadFont
- Axiom.hyperdoc.EmphasizeFont
- Axiom.hyperdoc.BoldFont
- Axiom.hyperdoc.Font

1.3 hypertex

```
Usage: hypertex [-s]
```

1.4 htsearch

Construct a page with a menu of references to the word. The syntax of the command is:

```
Usage: htsearch word
```

1.5 spadbuf

```
Usage: spadbuf page_name [completion_files]
```

1.6 hthits

```
Usage: hthits pattern htdb-file
```

1.7 ex2ht

```
Usage: ex2ht exfile.ht ...
```

1.8 htadd

HyperDoc database file manager

```
Usage: htadd [-s|-l|-f db-directory] [-d|-n] filenames
```

Chapter 2

The hypertex language

```
\$Data
\#
\%
\&
\,
\-
\/
\:
\[
\]
\_
\{
\}

\aleph
\aliascon#1#2
\aliascon{HomogeneousAggregate\&}{HOAGG-}
\allowbreak
\alpha
\argDef{"Axiom 2D"}
\asharp{}
\aspSectionNumber
\aspSectionTitle
\autobutt{BROWSEhelp}
\autobuttons
\autobuttLayout{\HelpButton{#1}}
\axiom
\axiom{  x +   y +   z = 8}
\axiomcommand{)lisp (defun f () (pprint "hello"))}
\axiomviewport
\axiomviewportasbutton
\axiomviewportbutton
```

```
\axiomxl{}
\axiomFunFromX
\axiomFunFrom{**}{Float}
\axiomFunX{declare}
\axiomFun{AND}
\axiomGlossSee{#1}{#2}
\axiomOpFrom{*}{QuadraticForm}
\axiomOpX
\axiomOp{#1!}
\axiomOp{*}
\axiomSig{Integer}{List Integer}
\axiomSyntax{()}
\axiomType{AbelianMonoid}

begin{array}{ccl} ... \end{array}

begin{page}{AlgebraPage}{Abstract Algebra}
   ...
\end{page}

\backslash
\baseLeftSkip
\baselineskip 10pt
\baselineskip
\beep

(Note: all begin and end items should be prefixed with a backslash)
begin{figXmpLines} ... end{figXmpLines}

begin{figure}[htbp] ... end{figure}

\beginImportant ... \endImportant

begin{items}[how wide am I] ... end{items}

begin{paste}{AssociationListXmpPageFull1}{AssociationListXmpPageEmpty1}
   ...
end{paste}

begin{patch}{AssociationListXmpPagePatch1}  ... end{patch}

begin{picture}(183,252)(-125,0) ... end{picture}

begin{quotation} ... end{quotation}

begin{scroll} ... end{scroll}

begin{spadsrc} ... end{spadsrc}

begin{tabular}{ccc} ... end{tabular}
```

```
begin{texonly} ... end{texonly}

begin{verbatim} ... end {verbatim}

begin{xmpLines} ... end{xmpLines}

\begingroup ... \endgroup
\beginitems ... \enditems
\beginmenu ... \endmenu
\beginscroll ... \endscroll

\bf
\bgroup
\bigbreak
\blankline
\bmod
\bot
\bound{Data}
\boxvalue{b1}
\Browse{}
\bs{}
\bullet

\caption{Three-Dimensional Lighting Panel.}
\cdot
\cdots
\center
\centerline
\chapref{ugPackages}.{ugCategories}{12.12.}{Anonymous Categories}  \Clef{}
\chi
\cite{gtz:gbpdpi}
\cleardoublepage
\command
\con
\conf
\controlbitmap
\ControlBitmap
\ControlBitmap{continue}
\coprod
\cos
\csch

\ddots
\def
\delta
\del
\displaystyle
\div
\dlink
```

```
\dom
\dot
\dots
\downarrow
\downlink{`Table'}{TableXmpPage}

\egroup
\ell
\else
\em
\emptyset
\end
\env{AXIOM}
\epsffile[0 0 295 295]{../ps/23dcola.ps}
\epsilon
\erf
\eta
\eth
\examplenumber
\exists
\ExitBitmap
\ExitBitmap{}
\exitbuttons
\ExitButton
\ExitButton{QuitPage}
\expr{1}
\exptypeindex{FortranCode}

\fakeAxiomFun{bubbleSort!}
\fbox{Boxed!}
\fi
\footnote
\forall
\frac{(x - 1)^2}{2}
\free{Data}
\frenchspacing
\funArgs{color}
\funSyntax{blue}

\Gallery{}
\gamma_{i,j}
\Gamma
\gdef
\generalFortranNumber
\generalFortranTitle
\geq
\glossSee
\gloss
\gotoevenpage
\GoBackToWork{}
```

```
\graphpaste{draw(cos(x*y),x=-3..3,y=-3..3)}

\hangafter=1
\hangindent=2pc
\hasreturn
\hbadness = 10001
\hbar
\hbox
\HDexptypeindex{Any}{ugTypesAnyNonePage}{2.6.}{The ``Any'' Domain}
\HDindex{list!association}{AssociationListXmpPage}{9.1}{AssociationList}
\HDsyscmdindex{abbreviation}{ugSysCmdcompilePage}{B.7.}{)compile}
\head{section}{Diversion: When Things Go Wrong}{ugIntProgDivTwo}
\head{subsection}{Arithmetic}{ugxCartenArith}
\helpbit{axes3D}
\HelpBitmap
\HelpBitmap{}
\HelpButton{#1}
\HelpButton{ugHyperPage}
\helppage{TestHelpPage}
\hidepaste
\horizontalline
\hspace
\htbitmap{f01qdf}
\htbitmap{great=}
\htbmdir{}
\htbmfile{pick}
\httex{At the end of the page}
\huge
\HyperName
\HyperName{}
\hyphenation

\ifcond
\ignore{Table}
\imath
\indent{0}
\indented
\indentrel{3}
\index{axiom}
\ind
\infty
\inputbitmap{/usr/include/X11/bitmaps/1x1}
\inputbox[1]{three}
\inputimage{{#1}.view/image}}
\inputpixmap
\inputstring{FindTopic}{15}{series}
\input{gallery/antoine.htex}
\int_{0}^{1}
\iota
\it
```

```
\item
\item[1. ]
\ixpt{}

\kappa
\keyword

\labelSpace{1.5pc}
\label{fig-clifalg}
\lambda
\Lambda
\lanb{}
\LangName
\Language{}
\large
\Large
\ldots
\le
\left
\leftarrow
\leq
\leqno(3)
\le
\lim_{x
\linebreak
\link
\Lisp{}
\lispcommand{Show Lisp definition}{(pprint (symbol-function 'HTXTESTPAGE))}
\lispdownlink{#1}{(|conPage| '|#2|)}
\lisplink{#1}{(|conOpPage| #2 '{#3})}}
\lispmemolink{Settings}{(|htSystemVariables|)}
\lispwindowlink{Link to it}{(HTXTESTPAGE "Hi there")}
\ll
\localinfo
\log

\mapsto
\marginpar
\mathOrSpad{1}
\mathop
\memolink{memolink to Actions menu}{HTXLinkTopPage}
\menudownlink{A Trigonometric Function of a Quadratic}{ExIntTrig}
\menuitemstyle{A.13. )history}}{ugSysCmdhistoryPage} }
\menulink{Number Theory}{NumberTheoryPage}
\menulispcommand{System Variables}{(|htsv|)}
\menulispdownlink{CO2AFF}{(|c02aff|)}
\menulispwindowlink{Browse}{(|kSearch| "NagEigenPackage")}
\menumemolink{AXIOM Book}{UsersGuidePage}
\menuspadref
\menuunixcmd
```

```
\menuunixcommand{Edit}{xterm}
\menuunixlink{Reference}
\menuunixwindow{Link}{cat
\menuwindowlink{About AXIOM}{RootPageLogo}
\menuxmpref{CliffordAlgebra}
\mid
\mu
\nabla
\nabla}{
\nagDocumentationNumber
\nagDocumentationTitle
\nagLinkIntroNumber
\nagLinkIntroTitle
\nagLinkUsageNumber
\nagLinkUsageTitle
\nagTechnicalNumber
\nagTechnicalTitle
\naglib{}
\narrowDisplay
\narrower
\neg
\newcommand{\aliasdom}[2]{\lispdownlink{#1}{(|conPage| '|#2|)}}
\newcommand{\autobuttLayout}[1]{\centerline{#1}}}
\newcommand{\autobuttMaker}[1]{\autobuttLayout{\HelpButton{#1}}}
\newcommand{\riddlebuttons}[1]{\autobuttLayout{\link{\HelpBitmap}{#1}}}
\newline
\newpage
\newsearchresultentry
\newspadclient}[1]{xterm -n "#1" -e
\noOutputXtc
\noindent
\nolimits
\nolines
\nonLibAxiomType{?(Integer)},
\nonfrenchspacing
\not=
\notequal
\nu
\nugNagdNumber
\nugNagdTitle
\nullXtc
\nullspadcommand

\off
\omega
\on
\ops
\optArg{option}
\outdent{Sierpinsky's Tetrahedron}
\over
```

```
\pagename
\pageref{fig-quadform}
\par
\parallel
\parindent=1pc
\partial
\pastebutton{AssociationListXmpPageFull1}{\hidepaste}
\pastecommand
\pastegraph
\phi
\Phi
\Phi_n
\pi
\Pi
\pm
\pp
\pred{i}
\prime
\prod
\protect
\Psi
\psXtc
\pspadfun{drawRibbons}
\pspadtype{DataList}

\quad

\radiobox[0]{rthree}{sample}
\raggedright
\ranb{}
\ref{fig-clifalg}.
\ReturnBitmap
\ReturnBitmap{}
\returnbutton{homebutton}{ReturnPage}
\rho
\riddlebuttons
\right
\rightarrow
\rm

\sc
\searchresultentry
\searchwindow{Start Search}
\setcounter{chapter}{0}}{}
\sf
\showBlurb{AssociationList}
\showpaste
\sigma
\Sigma
```

```
\sim
\simplebox
\sin
\sloppy
\small
\smath{(k,t)}
\sp
\space{-1}
\spad{al}
\spadatt{commutative("*")}
\spadcmd{)abbreviation query}
\spadcommand{Data := Record(monthsOld : Integer, gender : String)}
\spadFileExt{}
\spadfun{solve}
\spadfunX{concat}
\spadfunFrom{table}{AssociationList}
\spadfunFromX{delete}{AssociationList}
\spadgloss{Category} == T}
\spadglossSee{Conversion}{conversion}
\spadgraph{draw(besselI(alpha, 5), alpha = -12..12, unit==[5,20])}
\spadignore{e.g.}
\spadkey{Join}
\spadop{**}
\spadopFrom{**}{RadicalCategory}
\spadpaste{Data := Record(monthsOld : Integer, gender : String) \bound{Data}}
\spadref
\spadsig{(Integer,Integer)}{Fraction(Integer)}
\spadSyntax{and}
\spadsys{)cd}
\spadsyscom{)set function cache}
\spadtype{AssociationList}
\spadvar
\spadviewportasbutton{mobius}
\special{psfile=../ps/3dvolume.ps}
\sqrt{1-2 t z+t^2}
\StdExitButton{}
\StdHelpButton{}
\stringvalue{FindTopic}
\subscriptIt{color}{1}
\subscriptText{Float}{yOffset}
\subsubsection{Arithmetic}
\sum_{m=a}^{b}
\surd
\syscmdindex{set hyperdoc browse exposure}
\syscom
\s
\tab
\tab{0}
\tab{-2}
\table
```

```
\tan
\tau
\texbreak
\texht{$L^m_n (z)$}
\texnewline
\theta
\thispage
\threedim{}
\times
\tiny
\top
\triangle
\tt
\twodim{}
\typeout{check this example}



\tab{5}
\TeX{}
\texht{Gr\"{o}bner}{Groebner}
\texht{Poincar\'e}{Poincare
\Theta

\ugAppGraphicsNumber
\ugAppGraphicsTitle
\ugAvailCLEFNumber
\ugAvailCLEFTitle
\ugAvailSnoopNumber
\ugAvailSnoopTitle
\ugBrowseCapitalizationConventionNumber
\ugBrowseCapitalizationConventionTitle
\ugBrowseCrossReferenceNumber
\ugBrowseCrossReferenceTitle
\ugBrowseDescriptionPageNumber
\ugBrowseDescriptionPageTitle
\ugBrowseDomainButtonsNumber
\ugBrowseDomainButtonsTitle
\ugBrowseDomainNumber
\ugBrowseDomainTitle
\ugBrowseGivingParametersNumber
\ugBrowseGivingParametersTitle
\ugBrowseMiscellaneousFeaturesNumber
\ugBrowseMiscellaneousFeaturesTitle
\ugBrowseNumber
\ugBrowseOptionsNumber
\ugBrowseOptionsTitle
\ugBrowseStartNumber
\ugBrowseStartTitle
\ugBrowseTitle
```

```
\ugBrowseViewsOfConstructorsNumber
\ugBrowseViewsOfConstructorsTitle
\ugBrowseViewsOfOperationsNumber
\ugBrowseViewsOfOperationsTitle
\ugCategoriesAndPackagesNumber
\ugCategoriesAndPackagesTitle
\ugCategoriesAttributesNumber
\ugCategoriesAttributesTitle
\ugCategoriesAxiomsNumber
\ugCategoriesAxiomsTitle
\ugCategoriesConditionalsNumber
\ugCategoriesConditionalsTitle
\ugCategoriesCorrectnessNumber
\ugCategoriesCorrectnessTitle
\ugCategoriesDefaultsNumber
\ugCategoriesDefaultsTitle
\ugCategoriesDefsNumber
\ugCategoriesDefsTitle
\ugCategoriesDocNumber
\ugCategoriesDocTitle
\ugCategoriesExportsNumber
\ugCategoriesExportsTitle
\ugCategoriesHierNumber
\ugCategoriesHierTitle
\ugCategoriesMembershipNumber
\ugCategoriesMembershipTitle
\ugCategoriesNumber
\ugCategoriesParametersNumber
\ugCategoriesParametersTitle
\ugCategoriesTitle
\ugDomainsAddDomainNumber
\ugDomainsAddDomainTitle
\ugDomainsAssertionsNumber
\ugDomainsAssertionsTitle
\ugDomainsAssertionsTitle
\ugDomainsBrowseNumber
\ugDomainsBrowseTitle
\ugDomainsCliffordNumber
\ugDomainsCliffordTitle
\ugDomainsCreatingNumber
\ugDomainsCreatingTitle
\ugDomainsDataListsNumber
\ugDomainsDataListsTitle
\ugDomainsDatabaseConstructorNumber
\ugDomainsDatabaseConstructorTitle
\ugDomainsDatabaseNumber
\ugDomainsDatabaseTitle
\ugDomainsDefaultsNumber
\ugDomainsDefaultsTitle
\ugDomainsDefsNumber
```

```
\ugDomainsDefsTitle
\ugDomainsDemoNumber
\ugDomainsDemoTitle
\ugDomainsDemoTitle
\ugDomainsExamplesNumber
\ugDomainsExamplesTitle
\ugDomainsMultipleRepsNumber
\ugDomainsMultipleRepsTitle
\ugDomainsNumber
\ugDomainsOriginsNumber
\ugDomainsOriginsTitle
\ugDomainsPuttingNumber
\ugDomainsPuttingTitle
\ugDomainsQueryEquationsNumber
\ugDomainsQueryEquationsTitle
\ugDomainsQueryLanguageNumber
\ugDomainsQueryLanguageTitle
\ugDomainsRepNumber
\ugDomainsRepTitle
\ugDomainsShortFormsNumber
\ugDomainsShortFormsTitle
\ugDomainsTitle
\ugDomainsTitle
\ugDomsinsDatabaseNumber
\ugDomsinsDatabaseTitle
\ugFantoineNumber
\ugFantoineTitle
\ugFconformalNumber
\ugFconformalTitle
\ugFdhtriNumber
\ugFdhtriTitle
\ugFimagesEightNumber
\ugFimagesEightTitle
\ugFimagesFiveNumber
\ugFimagesFiveTitle
\ugFimagesFiveTitle
\ugFimagesOneNumber
\ugFimagesOneTitle
\ugFimagesSevenNumber
\ugFimagesSevenTitle
\ugFimagesSixNumber
\ugFimagesSixTitle
\ugFimagesThreeNumber
\ugFimagesThreeTitle
\ugFimagesTwoNumber
\ugFimagesTwoTitle
\ugFntubeNumber
\ugFntubeTitle
\ugFscherkNumber
\ugFscherkTitle
```

```
\ugFtetraNumber
\ugFtetraTitle
\ugFtknotNumber
\ugFtknotTitle
\ugGraphClipNumber
\ugGraphClipTitle
\ugGraphColorNumber
\ugGraphColorPaletteNumber
\ugGraphColorPaletteTitle
\ugGraphColorPaletteTitle
\ugGraphColorTitle
\ugGraphColorTitle
\ugGraphCoordNumber
\ugGraphCoordTitle
\ugGraphCoordTitle
\ugGraphMakeObjectNumber
\ugGraphMakeObjectTitle
\ugGraphNumber
\ugGraphThreeDBuildNumber
\ugGraphThreeDBuildTitle
\ugGraphThreeDControlNumber
\ugGraphThreeDControlTitle
\ugGraphThreeDNumber
\ugGraphThreeDOptionsNumber
\ugGraphThreeDOptionsTitle
\ugGraphThreeDOptionsTitle
\ugGraphThreeDParNumber
\ugGraphThreeDParTitle
\ugGraphThreeDParmNumber
\ugGraphThreeDParmTitle
\ugGraphThreeDPlotNumber
\ugGraphThreeDPlotTitle
\ugGraphThreeDTitle
\ugGraphThreeDopsNumber
\ugGraphThreeDopsTitle
\ugGraphTitle
\ugGraphTitle
\ugGraphTwoDControlNumber
\ugGraphTwoDControlTitle
\ugGraphTwoDNumber
\ugGraphTwoDOptionsNumber
\ugGraphTwoDOptionsTitle
\ugGraphTwoDOptionsTitle
\ugGraphTwoDParNumber
\ugGraphTwoDParTitle
\ugGraphTwoDPlaneNumber
\ugGraphTwoDPlaneTitle
\ugGraphTwoDPlotNumber
\ugGraphTwoDPlotTitle
\ugGraphTwoDTitle
```

```
\ugGraphTwoDappendNumber
\ugGraphTwoDappendTitle
\ugGraphTwoDappendTitle
\ugGraphTwoDbuildNumber
\ugGraphTwoDbuildTitle
\ugGraphTwoDbuildTitle
\ugGraphTwoDopsNumber
\ugGraphTwoDopsTitle
\ugHyperButtonsNumber
\ugHyperButtonsTitle
\ugHyperExampleNumber
\ugHyperExampleTitle
\ugHyperHeadingsNumber
\ugHyperHeadingsTitle
\ugHyperInputNumber
\ugHyperInputTitle
\ugHyperInputTitle
\ugHyperKeysNumber
\ugHyperKeysTitle
\ugHyperNumber
\ugHyperResourcesNumber
\ugHyperResourcesTitle
\ugHyperScrollNumber
\ugHyperScrollTitle
\ugHyperSearchNumber
\ugHyperSearchTitle
\ugHyperTitle
\ugHyperTitle
\ugInOutAlgebraNumber
\ugInOutAlgebraTitle
\ugInOutFortranNumber
\ugInOutFortranTitle
\ugInOutInNumber
\ugInOutInTitle
\ugInOutInTitle
\ugInOutNumber
\ugInOutOutNumber
\ugInOutOutTitle
\ugInOutScriptNumber
\ugInOutScriptTitle
\ugInOutSpadprofNumber
\ugInOutSpadprofTitle
\ugInOutTeXNumber
\ugInOutTeXTitle
\ugInOutTitle
\ugIntProgColorArrNumber
\ugIntProgColorArrTitle
\ugIntProgColorNumber
\ugIntProgColorTitle
\ugIntProgCompFunsNumber
```

```
\ugIntProgCompFunsTitle
\ugIntProgCompFunsTitle
\ugIntProgDrawingNumber
\ugIntProgDrawingTitle
\ugIntProgFunctionsNumber
\ugIntProgFunctionsTitle
\ugIntProgNewtonNumber
\ugIntProgNewtonTitle
\ugIntProgNumber
\ugIntProgPLCNumber
\ugIntProgPLCTitle
\ugIntProgRibbonNumber
\ugIntProgRibbonTitle
\ugIntProgTitle
\ugIntProgTitle
\ugIntProgVecFieldsNumber
\ugIntProgVecFieldsTitle
\ugIntroArithmeticNumber
\ugIntroArithmeticTitle
\ugIntroAssignNumber
\ugIntroAssignTitle
\ugIntroAssignTitle
\ugIntroCalcDerivNumber
\ugIntroCalcDerivTitle
\ugIntroCalcDerivTitle
\ugIntroCalcLimitsNumber
\ugIntroCalcLimitsTitle
\ugIntroCalcLimitsTitle
\ugIntroCallFunNumber
\ugIntroCallFunTitle
\ugIntroCallFunTitle
\ugIntroCollectNumber
\ugIntroCollectTitle
\ugIntroCommentsNumber
\ugIntroCommentsTitle
\ugIntroConversionNumber
\ugIntroConversionTitle
\ugIntroDiffEqnsNumber
\ugIntroDiffEqnsTitle
\ugIntroExpressionsNumber
\ugIntroExpressionsTitle
\ugIntroGraphicsNumber
\ugIntroGraphicsTitle
\ugIntroIntegrateNumber
\ugIntroIntegrateTitle
\ugIntroLongNumber
\ugIntroLongTitle
\ugIntroMacrosNumber
\ugIntroMacrosTitle
\ugIntroNumber
```

```
\ugIntroNumbersNumber
\ugIntroNumbersTitle
\ugIntroNumbersTitle
\ugIntroNumber
\ugIntroPreviousNumber
\ugIntroPreviousTitle
\ugIntroSeriesNumber
\ugIntroSeriesTitle
\ugIntroSeriesTitle
\ugIntroSolutionNumber
\ugIntroSolutionTitle
\ugIntroStartNumber
\ugIntroStartTitle
\ugIntroSysCmmandsNumber
\ugIntroSysCmmandsTitle
\ugIntroTitle
\ugIntroTitle
\ugIntroTwoDimNumber
\ugIntroTwoDimTitle
\ugIntroTwoDimTitle
\ugIntroTypesNumber
\ugIntroTypesTitle
\ugIntroTypoNumber
\ugIntroTypoTitle
\ugIntroTypoTitle
\ugIntroVariablesNumber
\ugIntroVariablesTitle
\ugIntroVariablesTitle
\ugIntroYouNumber
\ugIntroYouTitle
\ugLangAssignNumber
\ugLangAssignTitle
\ugLangAssignTitle
\ugLangBlocksNumber
\ugLangBlocksTitle
\ugLangBlocksTitle
\ugLangIfNumber
\ugLangIfTitle
\ugLangIfTitle
\ugLangItsNumber
\ugLangItsTitle
\ugLangItsTitle
\ugLangLoopsBreakMoreNumber
\ugLangLoopsBreakMoreTitle
\ugLangLoopsBreakNumber
\ugLangLoopsBreakTitle
\ugLangLoopsBreakTitle
\ugLangLoopsBreakVsNumber
\ugLangLoopsBreakVsTitle
\ugLangLoopsCompIntNumber
```

```
\ugLangLoopsCompIntTitle
\ugLangLoopsForInNMNumber
\ugLangLoopsForInNMSNumber
\ugLangLoopsForInNMSTitle
\ugLangLoopsForInNMTitle
\ugLangLoopsForInNNumber
\ugLangLoopsForInNTitle
\ugLangLoopsForInNumber
\ugLangLoopsForInPredNumber
\ugLangLoopsForInPredTitle
\ugLangLoopsForInPredTitle
\ugLangLoopsForInTitle
\ugLangLoopsForInTitle
\ugLangLoopsForInXLNumber
\ugLangLoopsForInXLTitle
\ugLangLoopsIterateNumber
\ugLangLoopsIterateTitle
\ugLangLoopsNumber
\ugLangLoopsParNumber
\ugLangLoopsParTitle
\ugLangLoopsReturnNumber
\ugLangLoopsReturnTitle
\ugLangLoopsReturnTitle
\ugLangLoopsTitle
\ugLangLoopsTitle
\ugLangLoopsWhileNumber
\ugLangLoopsWhileTitle
\ugLangNumber
\ugLangStreamsPrimesNumber
\ugLangStreamsPrimesTitle
\ugLangTitle
\ugLogicalSearchesNumber
\ugLogicalSearchesTitle
\ugPackagesAbstractNumber
\ugPackagesAbstractTitle
\ugPackagesAbstractTitle
\ugPackagesCapsulesNumber
\ugPackagesCapsulesTitle
\ugPackagesCompilingNumber
\ugPackagesCompilingTitle
\ugPackagesCondsNumber
\ugPackagesCondsTitle
\ugPackagesCondsTitle
\ugPackagesDomsNumber
\ugPackagesDomsTitle
\ugPackagesHowNumber
\ugPackagesHowTitle
\ugPackagesInputFilesNumber
\ugPackagesInputFilesTitle
\ugPackagesNamesNumber
```

```
\ugPackagesNamesTitle
\ugPackagesNumber
\ugPackagesPackagesNumber
\ugPackagesPackagesTitle
\ugPackagesParametersNumber
\ugPackagesParametersTitle
\ugPackagesSyntaxNumber
\ugPackagesSyntaxTitle
\ugPackagesTitle
\ugPackagesTitle
\ugProblemDEQNumber
\ugProblemDEQTitle
\ugProblemDEQTitle
\ugProblemEigenNumber
\ugProblemEigenTitle
\ugProblemEigenTitle
\ugProblemFactorAlgNumber
\ugProblemFactorAlgTitle
\ugProblemFactorFFNumber
\ugProblemFactorFFTitle
\ugProblemFactorIntRatNumber
\ugProblemFactorIntRatTitle
\ugProblemFactorNumber
\ugProblemFactorRatFunNumber
\ugProblemFactorRatFunTitle
\ugProblemFactorTitle
\ugProblemFactorTitle
\ugProblemFiniteNumber
\ugProblemFiniteTitle
\ugProblemFiniteTitle
\ugProblemGaloisNumber
\ugProblemGaloisTitle
\ugProblemGaloisTitle
\ugProblemGeneticNumber
\ugProblemGeneticTitle
\ugProblemIdealNumber
\ugProblemIdealTitle
\ugProblemIntegrationNumber
\ugProblemIntegrationTitle
\ugProblemIntegrationTitle
\ugProblemLaplaceNumber
\ugProblemLaplaceTitle
\ugProblemLimitsNumber
\ugProblemLimitsTitle
\ugProblemLimitsTitle
\ugProblemLinPolEqnNumber
\ugProblemLinPolEqnTitle
\ugProblemLinPolEqnTitle
\ugProblemNumber
\ugProblemNumericNumber
```

```
\ugProblemNumericTitle
\ugProblemNumericTitle
\ugProblemSeriesNumber
\ugProblemSeriesTitle
\ugProblemSeriesTitle
\ugProblemSymRootNumber
\ugProblemSymRootTitle
\ugProblemTitle
\ugSysCmdNumber
\ugSysCmdOverviewNumber
\ugSysCmdOverviewTitle
\ugSysCmdTitle
\ugSysCmdTitle
\ugSysCmdabbreviationNumber
\ugSysCmdabbreviationTitle
\ugSysCmdabbreviationTitle
\ugSysCmdbootNumber
\ugSysCmdbootTitle
\ugSysCmdbootTitle
\ugSysCmdcdNumber
\ugSysCmdcdTitle
\ugSysCmdcdTitle
\ugSysCmdclearNumber
\ugSysCmdclearTitle
\ugSysCmdclearTitle
\ugSysCmdcloseNumber
\ugSysCmdcloseTitle
\ugSysCmdcloseTitle
\ugSysCmdcompileNumber
\ugSysCmdcompileTitle
\ugSysCmdcompileTitle
\ugSysCmddisplayNumber
\ugSysCmddisplayTitle
\ugSysCmddisplayTitle
\ugSysCmdeditNumber
\ugSysCmdeditTitle
\ugSysCmdeditTitle
\ugSysCmdfinNumber
\ugSysCmdfinTitle
\ugSysCmdfinTitle
\ugSysCmdframeNumber
\ugSysCmdframeTitle
\ugSysCmdframeTitle
\ugSysCmdhelpNumber
\ugSysCmdhelpTitle
\ugSysCmdhistoryNumber
\ugSysCmdhistoryTitle
\ugSysCmdhistoryTitle
\ugSysCmdlibraryNumber
\ugSysCmdlibraryTitle
```

```
\ugSysCmdlibraryTitle
\ugSysCmdlispNumber
\ugSysCmdlispTitle
\ugSysCmdlispTitle
\ugSysCmdloadNumber
\ugSysCmdloadTitle
\ugSysCmdltraceNumber
\ugSysCmdltraceTitle
\ugSysCmdltraceTitle
\ugSysCmdpquitNumber
\ugSysCmdpquitTitle
\ugSysCmdpquitTitle
\ugSysCmdquitNumber
\ugSysCmdquitTitle
\ugSysCmdquitTitle
\ugSysCmdreadNumber
\ugSysCmdreadTitle
\ugSysCmdreadTitle
\ugSysCmdsetNumber
\ugSysCmdsetTitle
\ugSysCmdsetTitle
\ugSysCmdshowNumber
\ugSysCmdshowTitle
\ugSysCmdshowTitle
\ugSysCmdspoolNumber
\ugSysCmdspoolTitle
\ugSysCmdspoolTitle
\ugSysCmdsynonymNumber
\ugSysCmdsynonymTitle
\ugSysCmdsystemNumber
\ugSysCmdsystemTitle
\ugSysCmdsystemTitle
\ugSysCmdtraceNumber
\ugSysCmdtraceTitle
\ugSysCmdtraceTitle
\ugSysCmdundoNumber
\ugSysCmdundoTitle
\ugSysCmdundoTitle
\ugSysCmdwhatNumber
\ugSysCmdwhatTitle
\ugSysCmdwhatTitle
\ugTwoTwoAldorNumber
\ugTwoTwoAldorTitle
\ugTwoTwoCCLNumber
\ugTwoTwoCCLTitle
\ugTwoTwoHyperdocNumber
\ugTwoTwoHyperdocTitle
\ugTwoTwoNAGLinkNumber
\ugTwoTwoNAGLinkTitle
\ugTwoTwoPolynomialsNumber
```

```
\ugTwoTwoPolynomialsTitle
\ugTypesAnyNoneNumber
\ugTypesAnyNoneTitle
\ugTypesAnyNoneTitle
\ugTypesBasicDomainConsNumber
\ugTypesBasicDomainConsTitle
\ugTypesBasicDomainConsTitle
\ugTypesBasicNumber
\ugTypesBasicTitle
\ugTypesBasicTitle
\ugTypesConvertNumber
\ugTypesConvertTitle
\ugTypesConvertTitle
\ugTypesDeclareNumber
\ugTypesDeclareTitle
\ugTypesDeclareTitle
\ugTypesExposeNumber
\ugTypesExposeTitle
\ugTypesExposeTitle
\ugTypesNumber
\ugTypesPkgCallNumber
\ugTypesPkgCallTitle
\ugTypesPkgCallTitle
\ugTypesRecordsNumber
\ugTypesRecordsTitle
\ugTypesRecordsTitle
\ugTypesResolveNumber
\ugTypesResolveTitle
\ugTypesResolveTitle
\ugTypesSubdomainsNumber
\ugTypesSubdomainsTitle
\ugTypesSubdomainsTitle
\ugTypesTitle
\ugTypesTitle
\ugTypesUnionsNumber
\ugTypesUnionsTitle
\ugTypesUnionsTitle
\ugTypesUnionsWOSelNumber
\ugTypesUnionsWOSelTitle
\ugTypesUnionsWOSelTitle
\ugTypesUnionsWSelNumber
\ugTypesUnionsWSelTitle
\ugTypesWritingAbbrNumber
\ugTypesWritingAbbrTitle
\ugTypesWritingAbbrTitle
\ugTypesWritingModesNumber
\ugTypesWritingModesTitle
\ugTypesWritingModesTitle
\ugTypesWritingMoreNumber
\ugTypesWritingMoreTitle
```

```
\ugTypesWritingNumber
\ugTypesWritingOneNumber
\ugTypesWritingOneTitle
\ugTypesWritingTitle
\ugTypesWritingZeroNumber
\ugTypesWritingZeroTitle
\ugUserAnonDeclareNumber
\ugUserAnonDeclareTitle
\ugUserAnonExampNumber
\ugUserAnonExampTitle
\ugUserAnonNumber
\ugUserAnonTitle
\ugUserAnonTitle
\ugUserBlocksNumber
\ugUserBlocksTitle
\ugUserBlocksTitle
\ugUserCacheNumber
\ugUserCacheTitle
\ugUserCacheTitle
\ugUserCompIntNumber
\ugUserCompIntTitle
\ugUserCompIntTitle
\ugUserDatabaseNumber
\ugUserDatabaseTitle
\ugUserDecOpersNumber
\ugUserDecOpersTitle
\ugUserDecUndecNumber
\ugUserDecUndecTitle
\ugUserDeclareNumber
\ugUserDeclareTitle
\ugUserDeclareTitle
\ugUserDelayNumber
\ugUserDelayTitle
\ugUserDelayTitle
\ugUserFreeLocalNumber
\ugUserFreeLocalTitle
\ugUserFreeLocalTitle
\ugUserFunMacNumber
\ugUserFunMacTitle
\ugUserIntroNumber
\ugUserIntroTitle
\ugUserMacrosNumber
\ugUserMacrosTitle
\ugUserMacrosTitle
\ugUserMakeNumber
\ugUserMakeTitle
\ugUserMakeTitle
\ugUserNumber
\ugUserOneNumber
\ugUserOneTitle
```

```
\ugUserPalNumber
\ugUserPalTitle
\ugUserPieceBasicNumber
\ugUserPieceBasicTitle
\ugUserPieceBasicTitle
\ugUserPieceNumber
\ugUserPiecePickingNumber
\ugUserPiecePickingTitle
\ugUserPiecePredNumber
\ugUserPiecePredTitle
\ugUserPiecePredTitle
\ugUserPieceTitle
\ugUserRecurNumber
\ugUserRecurTitle
\ugUserRecurTitle
\ugUserRulesNumber
\ugUserRulesTitle
\ugUserRulesTitle
\ugUserTitle
\ugUserTitle
\ugUserTriangleNumber
\ugUserTriangleTitle
\ugUserTriangleTitle
\ugUserUseNumber
\ugUserUseTitle
\ugUserUseTitle
\ugWhatsNewAsharpNumber
\ugWhatsNewAsharpTitle
\ugWhatsNewDocumentationNumber
\ugWhatsNewDocumentationTitle
\ugWhatsNewHyperDocNumber
\ugWhatsNewHyperDocTitle
\ugWhatsNewImportantNumber
\ugWhatsNewImportantTitle
\ugWhatsNewLanguageNumber
\ugWhatsNewLanguageTitle
\ugWhatsNewLibraryNumber
\ugWhatsNewLibraryTitle
\ugWhatsNewNumber
\ugWhatsNewTitle
\ugWhatsNewTwoTwoNumber
\ugWhatsNewTwoTwoTitle
\ugXdefaultsNumber
\ugXdefaultsTitle
\ugxCliffordComplexNumber
\ugxCliffordComplexTitle
\ugxCliffordDiracNumber
\ugxCliffordDiracTitle
\ugxCliffordExteriorNumber
\ugxCliffordExteriorTitle
```

```
\ugxCliffordQuaternNumber
\ugxCliffordQuaternTitle
\ugxFactoredArithNumber
\ugxFactoredArithTitle
\ugxFactoredDecompNumber
\ugxFactoredDecompTitle
\ugxFactoredExpandNumber
\ugxFactoredExpandTitle
\ugxFactoredNewNumber
\ugxFactoredNewTitle
\ugxFactoredVarNumber
\ugxFactoredVarTitle
\ugxFloatConvertNumber
\ugxFloatConvertTitle
\ugxFloatHilbertNumber
\ugxFloatHilbertTitle
\ugxFloatHilbertTitle
\ugxFloatIntroNumber
\ugxFloatIntroTitle
\ugxFloatOutputNumber
\ugxFloatOutputTitle
\ugxIntegerBasicNumber
\ugxIntegerBasicTitle
\ugxIntegerNTNumber
\ugxIntegerNTTitle
\ugxIntegerPrimesNumber
\ugxIntegerPrimesTitle
\ugxLinearOrdinaryDifferentialOperatorOneRatNumber
\ugxLinearOrdinaryDifferentialOperatorOneRatTitle
\ugxLinearOrdinaryDifferentialOperatorSeriesNumber
\ugxLinearOrdinaryDifferentialOperatorSeriesTitle
\ugxLinearOrdinaryDifferentialOperatorTwoConstNumber
\ugxLinearOrdinaryDifferentialOperatorTwoConstTitle
\ugxLinearOrdinaryDifferentialOperatorTwoMatrixNumber
\ugxLinearOrdinaryDifferentialOperatorTwoMatrixTitle
\ugxListAccessNumber
\ugxListAccessTitle
\ugxListChangeNumber
\ugxListChangeTitle
\ugxListCreateNumber
\ugxListCreateTitle
\ugxListDotNumber
\ugxListDotTitle
\ugxListOtherNumber
\ugxListOtherTitle
\ugxMatrixCreateNumber
\ugxMatrixCreateTitle
\ugxMatrixOpsNumber
\ugxMatrixOpsTitle
\ugxProblemDEQSeriesNumber
```

```
\ugxProblemDEQSeriesTitle
\ugxProblemDEQSeriesTitle
\ugxProblemFiniteConversionNumber
\ugxProblemFiniteConversionTitle
\ugxProblemFiniteCyclicNumber
\ugxProblemFiniteCyclicTitle
\ugxProblemFiniteExtensionFiniteNumber
\ugxProblemFiniteExtensionFiniteTitle
\ugxProblemFiniteExtensionFiniteTitle
\ugxProblemFiniteModulusNumber
\ugxProblemFiniteModulusTitle
\ugxProblemFiniteNormalNumber
\ugxProblemFiniteNormalTitle
\ugxProblemFinitePrimeNumber
\ugxProblemFinitePrimeTitle
\ugxProblemFinitePrimeTitle
\ugxProblemFiniteUtilityNumber
\ugxProblemFiniteUtilityTitle
\ugxProblemFiniteUtilityTitle
\ugxProblemLDEQClosedNumber
\ugxProblemLDEQClosedTitle
\ugxProblemLinSysNumber
\ugxProblemLinSysTitle
\ugxProblemNLDEQClosedNumber
\ugxProblemNLDEQClosedTitle
\ugxProblemOnePolNumber
\ugxProblemOnePolTitle
\ugxProblemOnePolTitle
\ugxProblemPolSysNumber
\ugxProblemPolSysTitle
\ugxProblemPolSysTitle
\ugxProblemSeriesArithmeticNumber
\ugxProblemSeriesArithmeticTitle
\ugxProblemSeriesBernoulliNumber
\ugxProblemSeriesBernoulliTitle
\ugxProblemSeriesCoefficientsNumber
\ugxProblemSeriesCoefficientsTitle
\ugxProblemSeriesConversionsNumber
\ugxProblemSeriesConversionsTitle
\ugxProblemSeriesConversionsTitle
\ugxProblemSeriesCreateNumber
\ugxProblemSeriesCreateTitle
\ugxProblemSeriesFormulaNumber
\ugxProblemSeriesFormulaTitle
\ugxProblemSeriesFormulaTitle
\ugxProblemSeriesFunctionsNumber
\ugxProblemSeriesFunctionsTitle
\ugxProblemSeriesFunctionsTitle
\ugxProblemSeriesSubstituteNumber
\ugxProblemSeriesSubstituteTitle
```

```
\ugxProblemSymRootAllNumber
\ugxProblemSymRootAllTitle
\ugxProblemSymRootAllTitle
\ugxProblemSymRootOneNumber
\ugxProblemSymRootOneTitle
\undocumented
\unind
\unixcommand{(Postscript)}{ghostview
\unixlink{Some file}
\unixwindow
\uparrow$
\UpBitmap{}
\upbutton{Click here}{UpPage}
\upsilon
\Upsilon
\userfun{bubbleSort2}

\varphi
\vbox
\verb+--+
\vertline
\viewport{/tmp/mobius}
\viewportasbutton{/tmp/mobius}
\void{}
\vskip .5
\vskip 1pc
\vskip 4pt
\vspace
\vspace{-25}

\windowid
\windowlink{ErrorPage}{ErrorPage}

\xdefault{Bld14}
\Xi
\xtc{
This is a \pspadtype{Record} type with age and gender fields.
}{
\spadpaste{Data := Record(monthsOld : Integer, gender : String) \bound{Data}}
}
\xmpLine{)set fun comp on}{}

\zag{1}{6}+
\zeta
```

Chapter 3

Shared Code

3.0.1 BeStruct

⟨*BeStruct*⟩≡

```
typedef struct be_struct {
    int type;
    char *id;
    struct be_struct *next;
} BeStruct;

BeStruct *top_be_stack;
```

3.1 Shared Code for file handling

3.1.1 strpostfix

⟨*strpostfix*⟩≡

```
static int strpostfix(char *s, char *t) {
    int slen = strlen(s), tlen = strlen(t);
    if (tlen > slen)
        return 0;
    while (tlen > 0)
        if (s[--slen] != t[--tlen])
            return 0;
    return 1;
}
```

3.1.2 extendHT

If the filename does not end with the string ".pamphlet", or ".ht", or ".pht", then add ".ht" as the default. System pages live in the bookvol7.1.pamphlet file but user pages can live in .ht files. The .pht files are the "paste" files which are cached results of computations available when hyperdoc is running without Axiom.

For system pages we hand generate the paste files and add them to the hyperdoc volume.

⟨*extendHT*⟩≡

```
void extendHT(char *name) {
    if (!strpostfix(name, ".pamphlet") &&
        !strpostfix(name, ".ht") &&
        !strpostfix(name, ".pht"))
        strcat(name, ".ht");
    return;
}
```

3.1.3 buildHtFilename

This procedure is sent a filename, and from it tries to build the full filename, this it returns in the fullname variable. If the file is not found, then it returns a -1. The fname is the fullpath name for the file, including the .ht extension. The aname is the filename minus the added .ht extension, and the pathname.

⟨*buildHtFilename*⟩≡

```
static int buildHtFilename(char *fname, char *aname, char *name) {
    char cdir[256];
    char *c_dir;
    char *HTPATH;
    char *trace;
    char *trace2;
    int ht_file;
    if (cwd(name)) {
        /* user wants to use the current working directory */
        c_dir = (char *) getcwd(cdir, 254);
        strcpy(fname, c_dir);
        /* Now add the rest of the filename */
        strcat(fname, "/");
        strcat(fname, &name[2]);
        /** now copy the actual file name to addname **/
        for (trace = &name[strlen(name)]; trace != name &&
             (*trace != '/'); trace--);
        if (trace == name) {
            fprintf(stderr, "ht_open_file: Didn't expect a filename like %s\n",
                    name);
            exit(-1);
        }
        trace++;
        strcpy(aname, trace);

        /** add  the .ht extension if needed **/
        extendHT(aname);
        extendHT(fname);
       /*fprintf(stderr,
           "TPDHERE:ht_open_file:2: name=%s aname=%s fname=%s\n",
            name,aname,fname); */

        /* Now just try to access the file */
        return (access(fname, R_OK));
    }
    else if (pathname(name)) {
        /* filename already has the path specified */
        strcpy(fname, name);
```

```
        /** now copy the actual file name to addname **/
        for (trace = &name[strlen(name)]; trace != name &&
             (*trace != '/'); trace--);
        if (trace == name) {
            fprintf(stderr, "ht_open_file: Didn't expect a filename like %s\n",
                    name);
            exit(-1);
        }
        trace++;
        strcpy(aname, trace);

        /** add  the .ht extension if needed **/
        extendHT(aname);
        extendHT(fname);

        /* Now just try to access the file */
        return (access(fname, R_OK));
    }
    else {/** If not I am going to have to append path names to it **/
        HTPATH = (char *) getenv("HTPATH");
        if (HTPATH == NULL) {
        /** The user does not have a HTPATH, so I will use the the directory
        $AXIOM/doc as the default path ***/
          char *spad = (char *) getenv("AXIOM");
          if (spad == NULL) {
            fprintf(stderr,
            "htFileOpen:Cannot find ht data base: setenv HTPATH or AXIOM\n");
             exit(-1);
          }
          HTPATH = (char *) halloc(1024 * sizeof(char), "HTPATH");
          strcpy(HTPATH, spad);
          strcat(HTPATH, "/doc");
        }
        /** Now that I have filled HTPATH, I should try to open a file by the
          given name **/
        strcpy(aname, name);
        extendHT(aname);
        for (ht_file = -1, trace2 = HTPATH;
             ht_file == -1 && *trace2 != '\0';) {
            for (trace = fname; *trace2 != '\0' && (*trace2 != ':');)
                *trace++ = *trace2++;
            *trace++ = '/';
            *trace = 0;
            if (!strcmp(fname, "./")) {
                /** The person wishes me to check the current directory too **/
                getcwd(fname, 256);
```

```
                strcat(fname, "/");
            }
            if (*trace2)
                trace2++;
            strcat(fname, aname);
            ht_file = access(fname, R_OK);
        }
        return (ht_file);
    }
}
```

3.1.4 pathname

⟨*pathname*⟩≡

```
static int pathname(char *name) {
    while (*name)
        if (*name++ == '/')
            return 1;
    return 0;
}
```

3.1.5 htFileOpen

This procedure opens the proper HT file

⟨*htFileOpen*⟩≡

```
FILE *htFileOpen(char *fname, char *aname, char *name) {
    FILE *ht_fp;
    int ret_value;
    ret_value = buildHtFilename(fname, aname, name);
    if (ret_value == -1) {
        fprintf(stderr, "htFileOpen: Unknown file %s\n", fname);
        exit(-1);
    }
    ht_fp = fopen(fname, "r");
    if (ht_fp == NULL) {
        perror("htFileOpen");
        exit(-1);
    }
    return (ht_fp);
}
```

3.1.6 dbFileOpen

This function is responsible for actually opening the database file. For the moment it gets the $AXIOM environment variable, and appends to it "doc/ht.db", and then opens it

Modified on 12/3/89 to take a second argument. This argument tells the open routine whether it is reading the db file, or writing it. If writing is true, then I should check to insure I have proper write access. -JMW

Modified again on 12/9/89 so that it now uses HTPATH as the path name. Now it initially loads up the path name into a static variable. Then upon every trip, it gets the next ht.db found. It returns NULL when no ht.db is found. -JMW

⟨*dbFileOpen*⟩≡

```
FILE *dbFileOpen(char *dbFile) {
    static char *db_path_trace = NULL;
    char *dbFile_trace;
    FILE *db_fp;
    char *spad;
    /*
     * The first time through is the only time this could be true. If so, then
     * create the default HTPATH for gDatabasePath.
     */
    /*fprintf(stderr,"addfile:dbFileOpen: entered dbFile=%s\n",dbFile);*/
    if (gDatabasePath == NULL) {
        gDatabasePath = (char *) getenv("HTPATH");
        if (gDatabasePath == NULL) {
            spad = (char *) getenv("AXIOM");
            if (spad == NULL) {
                fprintf(stderr,
                   "addfile:dbFileOpen: Cannot find ht data base path:\n");
                exit(-1);
            }
            gDatabasePath = (char *) halloc(sizeof(char) * 1024, "dbFileOpen");
            strcpy(gDatabasePath, spad);
            strcat(gDatabasePath, "/doc");
        }
        db_path_trace = gDatabasePath;
    }
    /*fprintf(stderr,"addfile:dbFileOpen: db_path_trace=%s\n",db_path_trace);*/
    /*
     * Now Loop until I find one with okay filename
     */
    for (db_fp = NULL; db_fp == NULL && *db_path_trace != '\0';) {
        for (dbFile_trace = dbFile; *db_path_trace != ':' &&
             *db_path_trace != '\0'; db_path_trace++)
            *dbFile_trace++ = *db_path_trace;
```

```
        *dbFile_trace = '\0';
        strcat(dbFile_trace, "/ht.db");
/*    fprintf(stderr,"addfile:dbFileOpen: dbFile_trace=%s\n",dbFile_trace); */
/*    fprintf(stderr,"addfile:dbFileOpen: dbFile=%s\n",dbFile); */
        db_fp = fopen(dbFile, "r");
        if (*db_path_trace != '\0')
            db_path_trace++;
    }
/*
    if (db_fp == NULL)
      fprintf(stderr,"addfile:dbFileOpen: exit (null)\n");
    else
      fprintf(stderr,"addfile:dbFileOpen: exit opened\n");
*/
    return (db_fp);
}
```

3.1.7 tempFileOpen

⟨*tempFileOpen*⟩≡

```
FILE *tempFileOpen(char *temp_dbFile) {
    FILE *temp_db_fp;
    /** Just make the name and open it **/
    strcpy(temp_dbFile, temp_dir);
    strcat(temp_dbFile, "ht2.db" /* dbFileName */ );
    temp_db_fp = fopen(temp_dbFile, "w");
    if (temp_db_fp == NULL) {
        perror("tempFileOpen");
        exit(-1);
    }
    return temp_db_fp;
}
```

3.2 Shared Code for Hash Table Handling

3.2.1 halloc

Allocate memory and bomb if none left (HyperDoc alloc)

⟨*halloc*⟩≡

```
char *halloc(int bytes, char *msg) {
    static char buf[200];
    char *result;
#ifdef DEBUG
    static int first = 1;
    if (first) {
        fp = fopen("/tmp/hallocs", "w");
        first = 0;
    }
#endif
    result = (char *) malloc(bytes);
#ifdef DEBUG
    fprintf(fp, "%d\tAlocating %d Bytes for %s\n", result,bytes, msg);
#endif
    if (result == NULL) {
        sprintf(buf, "Ran out of memory allocating %s.\b", msg);
        fprintf(stderr, "%s\n", buf);
        exit(-1);
    }
    return result;
}
```

3.2.2 hashInit

Initialize a hash table.

⟨*hashInit*⟩≡

```
void hashInit(HashTable *table, int size, EqualFunction equal,
                HashcodeFunction hash_code) {
    int i;
    table->table =
        (HashEntry **) halloc(size * sizeof(HashEntry *), "HashEntry");
    for (i = 0; i < size; i++)
        table->table[i] = NULL;
    table->size = size;
    table->equal = equal;
    table->hash_code = hash_code;
    table->num_entries = 0;
}
```

3.2.3 freeHash

⟨*freeHash*⟩≡

```
void freeHash(HashTable *table, FreeFunction free_fun) {
  if (table) {
    int i;
    for (i = 0; i < table->size; i++) {
      HashEntry *e, *next;
      for (e = table->table[i]; e != NULL;) {
        next = e->next;
        (*free_fun) (e->data);
        (*e).data=0;
        free(e);
        e = next;
      }
    }
    free(table->table);
  }
}
```

3.2.4 hashInsert

Insert an entry into a hash table.

⟨*hashInsert*⟩≡

```
void hashInsert(HashTable *table, char *data, char *key) {
    HashEntry *entry = (HashEntry *) halloc(sizeof(HashEntry), "HashEntry");
    int code;
    entry->data = data;
    entry->key = key;
    code = (*table->hash_code) (key, table->size) % table->size;
#ifdef DEBUG
    fprintf(stderr, "Hash value = %d\n", code);
#endif
    entry->next = table->table[code];
    table->table[code] = entry;
    table->num_entries++;
}
```

3.2.5 hashFind

⟨*hashFind*⟩≡

```
char *hashFind(HashTable *table, char *key) {
    HashEntry *entry;
    int code = table->hash_code(key, table->size) % table->size;
    for (entry = table->table[code]; entry != NULL; entry = entry->next)
        if ((*table->equal) (entry->key, key))
            return entry->data;
    return NULL;
}
```

3.2.6 hashReplace

⟨*hashReplace*⟩≡

```
char *hashReplace(HashTable *table, char *data, char *key) {
    HashEntry *entry;
    int code = table->hash_code(key, table->size) % table->size;

    for (entry = table->table[code]; entry != NULL; entry = entry->next)
        if ((*table->equal) (entry->key, key)) {
            entry->data = data;
            return entry->data;
        }
    return NULL;
}
```

3.2.7 hashDelete

⟨*hashDelete*⟩≡

```
void hashDelete(HashTable *table, char *key) {
    HashEntry **entry;
    int code = table->hash_code(key, table->size) % table->size;
    for (entry = &table->table[code]; *entry != NULL; entry=&((*entry)->next))
        if ((*table->equal) ((*entry)->key, key)) {
            *entry = (*entry)->next;
            table->num_entries--;
            return;
        }
}
```

3.2.8 hashMap

⟨*hashMap*⟩≡

```
void hashMap(HashTable *table, MappableFunction func) {
    int i;
    HashEntry *e;
    if (table == NULL)
        return;
    for (i = 0; i < table->size; i++)
        for (e = table->table[i]; e != NULL; e = e->next)
            (*func) (e->data);
}
```

3.2.9 hashCopyEntry

⟨*hashCopyEntry*⟩≡

```
HashEntry *hashCopyEntry(HashEntry *e) {
    HashEntry *ne;
    if (e == NULL)
        return e;
    ne = (HashEntry *) halloc(sizeof(HashEntry), "HashEntry");
    ne->data = e->data;
    ne->key = e->key;
    ne->next = hashCopyEntry(e->next);
    return ne;
}

/* copy a hash table */
```

3.2.10 hashCopyTable

⟨*hashCopyTable*⟩≡

```
HashTable *hashCopyTable(HashTable *table) {
    HashTable *nt = (HashTable *) halloc(sizeof(HashTable), "copy hash table");
    int i;
    nt->size = table->size;
    nt->num_entries = table->num_entries;
    nt->equal = table->equal;
    nt->hash_code = table->hash_code;
    nt->table = (HashEntry **) halloc(nt->size * sizeof(HashEntry *),
                                      "copy table");
    for (i = 0; i < table->size; i++)
        nt->table[i] = hashCopyEntry(table->table[i]);
    return nt;
}
```

3.2.11 stringHash

Hash code function for strings.

⟨*stringHash*⟩≡

```
int stringHash(char *s, int size) {
    int c = 0;
    char *p =s;
    while (*p)
        c += *p++;
    return c % size;
}
```

3.2.12 stringEqual

Test strings for equality.

⟨*stringEqual*⟩≡

```
int stringEqual(char *s1, char *s2) {
    return (strcmp(s1, s2) == 0);
}
```

3.2.13 allocString

Make a fresh copy of the given string.

⟨*allocString*⟩≡

```
char *allocString(char *str) {
    char * result;
    result = halloc(strlen(str)+1,"String");
    strcpy(result,str);
    return (result);
}
```

3.3 Shared Code for Error Handling

3.3.1 jump

⟨*jump*⟩≡
```
void jump(void) {
    if (gWindow == NULL)
        exit(-1);
    longjmp(jmpbuf, 1);
    fprintf(stderr, "(HyperDoc) Long Jump failed, Exiting\n");
    exit(-1);
}
```

3.3.2 dumpToken

We need a function to print the token object for debugging.

To use this function the caller provides its own name and the token to be printed. For instance, a call would look like:

```
dumpToken("fnname",token)
```

There is no return value.

⟨*dumpToken*⟩≡
```
void dumpToken(char *caller, Token t) {
  fprintf(stderr,"TPDHERE:%s:dumpToken type=%s id=%s\n",
          caller,token_table[t.type],t.id);
}
```

3.3.3 printPageAndFilename

⟨*printPageAndFilename*⟩≡

```
void printPageAndFilename(void) {
    char obuff[128];
    if (gPageBeingParsed->type == Normal) {
        /*
         * Now try to inform the user as close to possible where the error
         * occurred
         */
        sprintf(obuff,
                "(HyperDoc) While parsing %s on line %d\n\tin the file %s\n",
                gPageBeingParsed->name, line_number,
                gPageBeingParsed->filename);
    }
    else if (gPageBeingParsed->type == SpadGen) {
        sprintf(obuff, "While parsing %s from the Spad socket\n",
                gPageBeingParsed->name);
    }
    else if (gPageBeingParsed->type == Unixfd) {
        sprintf(obuff, "While parsing %s from a Unixpipe\n",
                gPageBeingParsed->name);
    }
    else {
        /* Unknown page type */
        sprintf(obuff, "While parsing %s\n", gPageBeingParsed->name);
    }
    fprintf(stderr, "%s", obuff);
}
```

3.3.4 printNextTenTokens

⟨*printNextTenTokens*⟩≡

```
void printNextTenTokens(void) {
    int i;
    int v;
    fprintf(stderr, "Trying to print the next ten tokens\n");
    for (i = 0; i < 10; i++) {
        v = getToken();
        if (v == EOF)
            break;
        printToken();
    }
    fprintf(stderr, "\n");
}
```

3.3.5 printToken

Print out a token value.

⟨*printToken*⟩≡

```
void printToken(void) {
    if (token.type == Word)
        printf("%s ", token.id);
    else {
        tokenName(token.type);
        printf("\\%s ", ebuffer);
    }
    fflush(stdout);
}
```

3.3.6 tokenName

⟨*tokenName*⟩≡

```
void tokenName(int type) {
    if (type <= NumberUserTokens)
        strcpy(ebuffer, token_table[type]);
    else {
        switch (type) {
          case Lbrace:
            strcpy(ebuffer, "{");
            break;
          case Rbrace:
            strcpy(ebuffer, "}");
            break;
          case Macro:
            strcpy(ebuffer, token.id);
            break;
          case Group:
            strcpy(ebuffer, "{");
            break;
          case Pound:
            strcpy(ebuffer, "#");
            break;
          case Lsquarebrace:
            strcpy(ebuffer, "[");
            break;
          case Rsquarebrace:
            strcpy(ebuffer, "]");
            break;
          case Punctuation:
            strcpy(ebuffer, token.id);
            break;
          case Dash:
            strcpy(ebuffer, token.id);
            break;
          case Verbatim:
            strcpy(ebuffer, "\\begin{verbatim}");
            break;
          case Scroll:
            strcpy(ebuffer, "\\begin{scroll}");
            break;
          case Dollar:
            strcpy(ebuffer, "$");
            break;
          case Percent:
            strcpy(ebuffer, "%");
```

```
            break;
          case Carrot:
            strcpy(ebuffer, "^");
            break;
          case Underscore:
            strcpy(ebuffer, "_");
            break;
          case Tilde:
            strcpy(ebuffer, "~");
            break;
          case Cond:
            sprintf(ebuffer, "\\%s", token.id);
            break;
          case Icorrection:
            strcpy(ebuffer, "\\/");
            break;
          case Paste:
            strcpy(ebuffer, "\\begin{paste}");
            break;
          case Patch:
            strcpy(ebuffer, "\\begin{patch}");
            break;
          default:
            sprintf(ebuffer, " %d ", type);
        }
        /*return 1;*/
    }
}
```

3.3.7 htperror

This is the error handling routine in AXIOM. The main routine is called htperror(): arguments: msg - like perror it accepts an error message to be printed errno - the errno which occurred. This is so an appropriate error message can be printed.

The prints out the page name, and then the filename in which the error occurred. If possible it also tries to print out the next ten tokens.

⟨*htperror*⟩≡

```
void tpderror(char *msg, int errn) {
    char obuff[256];
    /* The first thing I do is create the error message */
    if (errno <= Numerrors) {
        sprintf(obuff, "%s:%s\n", msg, errmess[errno]);
    }
    else {
        sprintf(obuff, "%s:\n", msg);
        fprintf(stderr, "Unknown error type %d\n", errno);
    }
    fprintf(stderr, "%s", obuff);
    printPageAndFilename();
    printNextTenTokens();
}
```

3.4 Shared Code for Lexical Analyzer

Lexical analyzer stuff. Exported functions:

- parserInit() – initialize the parser tables with keywords
- initScanner() – initialize scanner for reading a new page
- getToken() – sets the "token" variable to be the next token in the current input stream
- saveScannerState() – save the current state of scanner so that the scanner input mode may be switched
- restoreScannerState() – undo the saved state

Note: The scanner reads from four separate input locations depending on the value of the variable "inputType". If this variable is:

- FromFile – it read from the file pointed to by "cfile".
- FromString – It reads from the string "inputString".
- FromSpadSocket – It reads from the socket pointed to by spadSocket
- FromFD – It reads from a file descriptor

3.4.1 parserInit

Initialize the parser keyword hash table.

⟨*parserInit*⟩≡

```
void parserInit(void) {
    int i;
    Token *toke;
    /* First I initialize the hash table for the tokens */
    hashInit(
              &tokenHashTable,
              TokenHashSize,
              (EqualFunction)stringEqual,
              (HashcodeFunction)stringHash);
    for (i = 2; i <= NumberUserTokens; i++) {
        toke = (Token *) halloc(sizeof(Token), "Token");
        toke->type = i;
        toke->id = token_table[i];
        hashInsert(&tokenHashTable, (char *)toke, toke->id);
    }
}
```

3.4.2 initScanner

Initialize the lexical scanner to read from a file.

⟨*initScanner*⟩≡

```
void initScanner(void) {
    if (getenv("HTASCII")) {
        useAscii = (strcmp(getenv("HTASCII"), "yes") == 0);
    }
    else {
        if(gTtFontIs850==1) useAscii = 0;
        else useAscii = 1;
    }
    keyword = 0;
    last_ch = NoChar;
    last_token = 0;
    inputType = FromFile;
    fpos = 0;
    keyword_fpos = 0;
    last_command = -1;
    line_number = 1;
}
```

3.4.3 saveScannerState

These variables save the current state of scanner. Currently only one level of saving is allowed. In the future we should allow nested saves.

⟨*saveScannerState*⟩≡

```
void saveScannerState(void) {
    StateNode *new_item=(StateNode *)halloc((sizeof(StateNode)), "StateNode");
    new_item->page_start_fpos = page_start_fpos;
    new_item->fpos = fpos;
    new_item->keyword_fpos = keyword_fpos;
    new_item->last_ch = last_ch;
    new_item->last_token = last_token;
    new_item->token = token;
    new_item->inputType = inputType;
    new_item->inputString = inputString;
    new_item->cfile = cfile;
    new_item->next = top_state_node;
    new_item->keyword = keyword;
    top_state_node = new_item;
}
```

3.4.4 restoreScannerState

Restore the saved scanner state.

⟨*restoreScannerState*⟩≡

```
void restoreScannerState(void) {
    StateNode *x = top_state_node;
    if (top_state_node == NULL) {
        fprintf(stderr, "Restore Scanner State: State empty\n");
        exit(-1);
    }
    top_state_node = top_state_node->next;
    page_start_fpos = x->page_start_fpos;
    fpos = x->fpos;
    keyword_fpos = x->keyword_fpos;
    last_ch = x->last_ch;
    last_token = x->last_token;
    token = x->token;
    inputType = x->inputType;
    inputString = x->inputString;
    cfile = x->cfile;
    keyword = x->keyword;
    if (cfile != NULL)
        fseek(cfile, fpos + page_start_fpos, 0);
    /** Once that is done, lets throw away some memory **/
    free(x);
}
```

3.4.5 ungetChar

Return the character to the input stream.

⟨*ungetChar*⟩≡

```
void ungetChar(int c) {
    if (c == '\n')
        line_number--;
    last_ch = c;
}
```

3.4.6 getChar

⟨*getChar*⟩≡

```
int getChar(void) {
    int c;
    c = getChar1();
    if (useAscii) {
        switch (c) {
          case '':
            c = '-';
            break;
          case '':
            c = '+';
            break;
          case '':
            c = '[';
            break;
          case '':
            c = '+';
            break;
          case '':
            c = '-';
            break;
          case '':
            c = '+';
            break;
          case '':
            c = '-';
            break;
          case '':
            c = '+';
            break;
          case '':
            c = ']';
            break;
          case '':
            c = '+';
            break;
          case '':
            c = '|';
            break;
          default:
            break;
        }
    }
    return c;
```

```
}
```

3.4.7 getChar1

Return the next character in the input stream.

⟨*getChar1*⟩≡

```
static int getChar1(void) {
    int c;
    int cmd;
    if (last_ch != NoChar) {
        c = last_ch;
        last_ch = NoChar;
        if (c == '\n')
            line_number++;
        return c;
    }
    switch (inputType) {
      case FromUnixFD:
        c = getc(unixfd);
        if (c == '\n')
            line_number++;
        return c;
      case FromString:
        c = (*inputString ? *inputString++ : EOF);
        if (c == '\n')
            line_number++;
        return c;
      case FromFile:
        c = getc(cfile);
        fpos++;
        if (c == '\n')
            line_number++;
        return c;
      case FromSpadSocket:
AGAIN:
        if (*inputString) {
            /* this should never happen for the first character */
            c = *inputString++;
            if (c == '\n')
                line_number++;
            return c;
        }
        if (last_command == EndOfPage)
            return EOF;
        if (read_again == NULL) {
            last_command = cmd = get_int(spadSocket);
            if (cmd == EndOfPage)
```

```
                return EOF;
#ifndef HTADD
            if (cmd == SpadError)
                spadErrorHandler();
#endif
        }
        read_again = get_string_buf(spadSocket, sock_buf, 1023);
        /* this will be null if this is the last time*/
        inputString = sock_buf;
        goto AGAIN;
      default:
        fprintf(stderr, "Get Char: Unknown type of input: %d\n", inputType);
        return -1;
    }
}
```

3.4.8 ungetToken

Return current token to the input stream.

⟨*ungetToken*⟩≡

```
void ungetToken(void) {
    last_token = 1;
    unget_toke.type = token.type;
    unget_toke.id = allocString(token.id - 1);
}
```

3.4.9 getToken

⟨*getToken*⟩≡

```
int getToken(void) {
    int c, ws;
    int nls = 0;
    static int seen_white = 0;
    static char buffer[1024];
    char *buf = buffer;
    if (last_token) {
        last_token = 0;
        token.type = unget_toke.type;
        strcpy(buffer, unget_toke.id);
        free(unget_toke.id);
        token.id = buffer + 1;
        if (token.type == EOF)
            return EOF;
        else
            return 0;
    }
    seen_white = nls = 0;
    do {
        c = getChar();
        ws = whitespace(c);
        if (ws)
            seen_white++;
        if (c == '\n') {
            if (nls) {
                token.type = Par;
                return 0;
            }
            else
                nls++;
        }
    } while (ws);
    /* first character of string indicates number of spaces before token */
    if (!keyword)
        *buf++ = seen_white;
    else
        *buf++ = 0;
    keyword = 0;
    if (inputType != FromSpadSocket && c == '%') {
        while ((c = getChar()) != '\n' && c != EOF);
    /* trying to fix the comment problem: a comment line forces words
       on either side together*/
    /* try returning the eol */
```

```
    ungetChar(c);
    return getToken();
}
if (inputType == FromFile && c == '$') {
    token.type = Dollar;
    return 0;
}
switch (c) {
  case EOF:
    token.type = -1;
    return EOF;
  case '\\':
    keyword_fpos = fpos - 1;
    c = getChar();
    if (!isalpha(c)) {
        *buf++ = c;
        token.type = Word;
        *buf = '\0';
        seen_white = 0;
    }
    else {
        do {
            *buf++ = c;
        } while ((c = getChar()) != EOF && isalpha(c));

        ungetChar(c);
        *buf = '\0';
        keyword = 1;
        token.id = buffer + 1;
        return (keywordType());
    }
    break;
  case '{':
    token.type = Lbrace;
    break;
  case '}':
    token.type = Rbrace;
    break;
  case '[':
    token.type = Lsquarebrace;
    *buf++ = c;
    *buf = '\0';
    token.id = buffer + 1;
    break;
  case ']':
    token.type = Rsquarebrace;
```

```
      *buf++ = c;
      *buf = '\0';
      token.id = buffer + 1;
      break;
    case '#':
      token.type = Pound;
      /*
       * if I get a pound then what I do is parse until I get something
       * that is not an integer
       */
      c = getChar();
      while (isdigit(c) && (c != EOF)) {
          *buf++ = c;
          c = getChar();
      }
      ungetChar(c);
      *buf = '\0';
      token.id = buffer + 1;
      break;
    case '`':
    case '\'':
    case ',':
    case '.':
    case '!':
    case '?':
    case '"':
    case ':':
    case ';':
      token.type = Punctuation;
      *buf++ = c;
      *buf = '\0';
      /** Now I should set the buffer[0] as my flag for whether I had
        white-space in front of me, and whether I had white space
        behind me **/
      if (buffer[0])
          buffer[0] = FRONTSPACE;
      c = getChar();
      if (whitespace(c))
          buffer[0] |= BACKSPACE;
      ungetChar(c);
      token.id = buffer + 1;
      break;
    case '-':
      do {
          *buf++ = c;
      } while (((c = getChar()) != EOF) && (c == '-'));
```

```
        ungetChar(c);
        *buf = '\0';
        token.type = Dash;
        token.id = buffer + 1;
        break;
      default:
        do {
            *buf++ = c;
        } while ((c = getChar()) != EOF && !delim(c));
        ungetChar(c);
        *buf = '\0';
        token.type = Word;
        token.id = buffer + 1;
        break;
    }
    // dumpToken("getToken",token);
    return 0;
}
```

3.4.10 pushBeStack

⟨*pushBeStack*⟩≡

```
void pushBeStack(int type,char * id) {
    BeStruct *be = (BeStruct *) halloc(sizeof(BeStruct), "BeginENd stack");
    if (gWindow != NULL) {
        be->type = type;
        be->next = top_be_stack;
        be->id = allocString(id);
        top_be_stack = be;
    }
    return;
}
```

3.4.11 checkAndPopBeStack

This routine pops the be stack and compares types. If they are the same then I am okay and return a 1. Else I return a two and try to print a meaningful message.

⟨*checkAndPopBeStack*⟩≡

```
void checkAndPopBeStack(int type,char * id) {
    BeStruct *x;
    if (gWindow == NULL)
        return;
    if (top_be_stack == NULL) { /* tried to pop when I shouldn't have */
        fprintf(stderr, "Unexpected \\end{%s} \n", token.id);
        printPageAndFilename();
        printNextTenTokens();
        jump();
    }
    x = top_be_stack;
    if (x->type == type) {
        top_be_stack = top_be_stack->next;
        free(x->id);
        free(x);
        return;
    }
    /* else I didn't have a match. Lets try to write a sensible message */
    fprintf(stderr, "\\begin{%s} ended with \\end{%s} \n", x->id, id);
    printPageAndFilename();
    printNextTenTokens();
    jump();
}
```

3.4.12 clearBeStack

⟨*clearBeStack*⟩≡

```
int clearBeStack(void) {
    BeStruct *x = top_be_stack, *y;
    top_be_stack = NULL;
    while (x != NULL) {
        y = x->next;
        free(x);
        x = y;
    }
    return 1;
}
```

3.4.13 beType

⟨*beType*⟩≡

```
int beType(char *which) {
    Token store;
    getExpectedToken(Lbrace);
    getExpectedToken(Word);
    switch (token.id[0]) {
      case 't':
        if (!strcmp(token.id, "titems")) {
            token.type = Begintitems;
        }
        else {
            return -1;
        }
        break;
      case 'p':
        if (!strcmp(token.id, "page")) {
            token.type = Page;
        }
        else if (!strcmp(token.id, "paste")) {
            token.type = Paste;
        }
        else if (!strcmp(token.id, "patch")) {
            token.type = Patch;
        }
        else {
            return -1;
        }
        break;
      case 'v':         /* possibly a verbatim mode */
        if (!strcmp(token.id, "verbatim")) {
            token.type = Verbatim;
        }
        else {
            return -1;
        }
        break;
      case 's':         /* possibly a scroll mode */
        if (!strcmp("scroll", token.id)) {
            token.type = Beginscroll;
        }
        else if (!strcmp(token.id, "spadsrc")) {
            token.type = Spadsrc;
        }
        else {
```

```
            return -1;
        }
        break;
      case 'i':         /* possibly a item */
        if (!strcmp("items", token.id)) {
            token.type = Beginitems;
        }
        else {
            return -1;
        }
        break;
      default:
        return -1;
    }
    store.type = token.type;
    /* store.id = allocString(token.id); */
    getExpectedToken(Rbrace);
    token.type = store.type;

    /*
     * strcpy(token.id, store.id); free(store.id);
     */
    return 0;
}
```

3.4.14 beginType

This routine parses a statement of the form `\begin{word}`. Once it has read the word it tries to assign it a type. Once that is done it sends the word id, and the type to pushBeStack and then returns the type. For the moment I cannot even going to use a hashTable, although in the future this may be needed.

⟨*beginType*⟩≡

```
int beginType(void) {
    /*Token store;*/
    int ret_val;
    ret_val = beType("begin");
    if (ret_val == -1) {
        if (gWindow == NULL || gInVerbatim)
            return 1;
        else {
            fprintf(stderr, "Unknown begin type \\begin{%s} \n", token.id);
            printPageAndFilename();
            printNextTenTokens();
            jump();
        }
    }
    else {
        if (gWindow != NULL && !gInVerbatim && token.type != Verbatim
            && token.type != Spadsrc) {
            /* Now here I should push the needed info and then get */
            pushBeStack(token.type, token.id);
        }
        return 1;
    }
    return 1;
}
```

3.4.15 endType

This routine gets the end type just as the beginType routine does, But then it checks to see if recieved the proper endType. By a clever trick, the proper end type is 3000 + type. When environments this will have to change.

⟨*endType*⟩≡

```
int endType(void) {
    int ret;
    ret = beType("end");
    if (ret == -1) {
        /* unrecognized end token */
        if (gWindow == NULL || gInVerbatim) {
            return 1;
        }
        else {
            fprintf(stderr, "Unknown begin type \\begin{%s} \n", token.id);
            printPageAndFilename();
            printNextTenTokens();
            jump();
        }
    }
    else {
        if (gWindow != NULL && !gInVerbatim) {
            checkAndPopBeStack(token.type, token.id);
            token.type += 3000;
            return 1;
        }
        else {
            if (gWindow != NULL && ((gInVerbatim && token.type == Verbatim) ||
                                    (gInSpadsrc && token.type == Spadsrc))) {
                checkAndPopBeStack(token.type, token.id);
                token.type += 3000;
                return 1;
            }
            else {
                token.type += 3000;
                return 1;
            }
        }
    }
    return 1;
}
```

3.4.16 keywordType

⟨*keywordType*⟩≡

```
int keywordType(void) {
    Token *token_ent;
    /* first check to see if it is a reserved token */
    token_ent = (Token *) hashFind(&tokenHashTable, token.id);
    if (token_ent != NULL) {
        token.type = token_ent->type;

        /*
         * if I am a keyword I also have to check to see if I am a begin or
         * an end
         */
        if (token.type == Begin)
            return beginType();
        if (token.type == End)
            return endType();
        /* next check to see if it is a macro */
    }
    else if (gWindow != NULL) {
        if (hashFind(gWindow->fMacroHashTable, token.id) != NULL)
            token.type = Macro;
        else if (gPageBeingParsed->box_hash != NULL &&
                 hashFind(gPageBeingParsed->box_hash, token.id) != NULL)
        {
            token.type = Boxcond;
        }
        else if (hashFind(gWindow->fCondHashTable, token.id) != NULL)
            token.type = Cond;
        else                    /* We have no idea what we've got */
            token.type = Unkeyword;
    }
    else {                      /* We am probably in htadd so just return. It
                                 * is only concerned with pages anyway */
        token.type = Unkeyword;
    }
    return 0;
}
```

3.4.17 getExpectedToken

Read a token, and report a syntax error if it has the wrong type.

⟨*getExpectedToken*⟩≡

```
void getExpectedToken(int type) {
    getToken();
    if (token.type != type) {
        tokenName(type);
        fprintf(stderr, "syntax error: expected a %s\n", ebuffer);
        if (token.type == EOF) {
            printPageAndFilename();
            fprintf(stderr, "Unexpected EOF\n");
        }
        else {
            tokenName(token.type);
            fprintf(stderr, "not a %s\n", ebuffer);
            printPageAndFilename();
            printNextTenTokens();
        }
        longjmp(jmpbuf, 1);
        fprintf(stderr, "Could not jump to Error Page\n");
        exit(-1);
    }
}
```

3.4.18 spadErrorHandler

⟨*spadErrorHandler*⟩≡

```
static void spadErrorHandler(void) {
    /* fprintf(stderr, "got a spad error\n"); */
    longjmp(jmpbuf, 1);
    fprintf(stderr, "(HyperDoc) Fatal Error: Could not jump to Error Page.\n");
    exit(-1);
}
```

3.4.19 resetConnection

⟨*resetConnection*⟩≡

```
void resetConnection(void) {
    if (spadSocket) {
        FD_CLR(spadSocket->socket, &socket_mask);
        purpose_table[spadSocket->purpose] = NULL;
        close(spadSocket->socket);
        spadSocket->socket = 0;
        spadSocket = NULL;
        if (inputString)
            inputString[0] = '\0';
        read_again = 0;
        str_len = 0;
        still_reading = 0;
        connectSpad();
    }
}
```

3.4.20 spadBusy

Returns true if spad is currently computing.

⟨*spadBusy*⟩≡

```
int spadBusy(void) {
    if (sessionServer == NULL)
        return 1;
    send_int(sessionServer, QuerySpad);
    return get_int(sessionServer);
}

/* connect to AXIOM , return 0 if succesful, 1 if not */
```

3.4.21 connectSpad

⟨*connectSpad*⟩≡

```
int connectSpad(void) {
    if (!MenuServerOpened) {
        fprintf(stderr,"(HyperDoc) Warning: Not connected to AXIOM Server!\n");
        LoudBeepAtTheUser();
        return NotConnected;
    }
    if (spadSocket == NULL) {
        spadSocket = connect_to_local_server(SpadServer, MenuServer, Forever);
        if (spadSocket == NULL) {
            fprintf(stderr,
                   "(HyperDoc) Warning: Could not connect to AXIOM Server!\n");
            LoudBeepAtTheUser();
            return NotConnected;
        }
    }
    /* if (spadBusy()) return SpadBusy; */
    return Connected;
}
```

3.5 htadd shared code

⟨*htadd shared code*⟩≡

```
#include "bsdsignal.h"
#include "bsdsignal.h1"
#include "sockio-c.h1"

#define cwd(n) ((n[0] == '.' && n[1] == '/')?(1):(0))
#define TokenHashSize   100

FILE *cfile;                    /* currently active file pointer */

char ebuffer[128];

long fpos;                      /* Position of pointer in file in characters */

short int gInSpadsrc = 0;
short int gInVerbatim;
HyperDocPage *gPageBeingParsed;

char *inputString;             /* input string read when from_string is true */
int inputType;                 /* indicates where to read input */

jmp_buf jmpbuf;

int keyword;                  /* the last command was a keyword, or a group */
long keyword_fpos;              /* fpos of beginning of most recent keyword */

int last_ch;                    /* last character read, for ungetChar */
int last_command;               /* the last socket command */
int last_token;                 /* most recently read token for ungetToken */
int line_number;

long page_start_fpos;           /* where the current pages fpos started      */

char *read_again = 0;

char sock_buf[1024];            /* buffer for socket input */

Token token;                    /* most recently read token */
static HashTable tokenHashTable;           /* hash table of parser tokens */
StateNode *top_state_node;
Token unget_toke;

FILE *unixfd;
int useAscii; /* should we translate graphics characters on the fly */
```

```
void printPageAndFilename(void);
void printNextTenTokens(void);

extern char *token_table[];

char *token_table[] = {
  "",           /* Dummy token name */
  "word",
  "page",
  "lispcommandquit",
  "bf",
  "link",
  "downlink",
  "beginscroll",
  "spadcommand",
  "nolines",
  "env",
  "par",
  "centerline",
  "begin",
  "beginitems",
  "item",
  "table",
  "fbox",
  "tab",
  "space",
  "indent",
  "horizontalline",
  "newline",
  "enditems",
  "returnbutton",
  "memolink",
  "upbutton",
  "endscroll",
  "thispage",
  "returnto",
  "free",
  "bound",
  "lisplink",
  "unixlink",
  "mbox",
  "inputstring",
  "stringvalue",
  "spadlink",
  "inputbitmap",
```

```
"inputpixmap",
"unixcommand",
"em",
"lispcommand",
"lispmemolink",
"lispdownlink",
"spadcall",
"spadcallquit",
"spaddownlink",
"spadmemolink",
"qspadcall",
"qspadcallquit",
"inputbox",
"radioboxes",
"boxvalue",
"vspace",
"hspace",
"newcommand",
"windowid",
"beep",
"quitbutton",
"begintitems",
"titem",
"end",
"it",
"sl",
"tt",
"rm",
"ifcond",
"else",
"fi",
"newcond",
"setcond" ,
"button",
"windowlink",
"haslisp",
"hasup",
"hasreturn",
"hasreturnto",
"lastwindow",
"endtitems",
"lispwindowlink",
"beginpile",
"endpile",
"nextline",
"pastebutton",
```

```
"color",
"helppage",
"patch",
"radiobox",
"ifrecond",
"math",
"mitem",
"pagename",
"examplenumber",
"replacepage",
"inputimage",
"spadgraph",
"indentrel",
"controlbitmap"
};
```

⟨*token.h*⟩
⟨*spadErrorHandler*⟩
⟨*spadBusy*⟩
⟨*connectSpad*⟩
⟨*resetConnection*⟩
⟨*pathname*⟩
⟨*BeStruct*⟩
⟨*strpostfix*⟩
⟨*extendHT*⟩
⟨*buildHtFilename*⟩
⟨*htFileOpen*⟩
⟨*tempFileOpen*⟩
⟨*halloc*⟩
⟨*hashInit*⟩
⟨*hashInsert*⟩
⟨*hashDelete*⟩
⟨*hashMap*⟩
⟨*hashFind*⟩
⟨*hashReplace*⟩
⟨*freeHash*⟩
⟨*stringHash*⟩
⟨*stringEqual*⟩
⟨*allocString*⟩
⟨*jump*⟩
⟨*tokenName*⟩
⟨*printToken*⟩
⟨*printPageAndFilename*⟩
⟨*printNextTenTokens*⟩
⟨*parserInit*⟩
⟨*initScanner*⟩

⟨*saveScannerState*⟩
⟨*restoreScannerState*⟩
⟨*ungetChar*⟩
⟨*getExpectedToken*⟩
⟨*ungetToken*⟩
⟨*getChar1*⟩
⟨*getChar*⟩
⟨*getToken*⟩
⟨*pushBeStack*⟩
⟨*clearBeStack*⟩
⟨*checkAndPopBeStack*⟩
⟨*beType*⟩
⟨*beginType*⟩
⟨*endType*⟩
⟨*keywordType*⟩

3.6 hypertex shared code

⟨*hypertex shared code*⟩≡

```
#include "bsdsignal.h"
#include "bsdsignal.h1"
#include "sockio-c.h1"

#define cwd(n) ((n[0] == '.' && n[1] == '/')?(1):(0))
#define TokenHashSize   100

FILE *cfile;                    /* currently active file pointer */

char ebuffer[128];

long fpos;                      /* Position of pointer in file in characters */

short int gInSpadsrc = 0;
short int gInVerbatim;
HyperDocPage *gPageBeingParsed;

char *inputString;             /* input string read when from_string is true */
int inputType;                 /* indicates where to read input */

jmp_buf jmpbuf;

int keyword;                  /* the last command was a keyword, or a group */
long keyword_fpos;              /* fpos of beginning of most recent keyword */

int last_ch;                    /* last character read, for ungetChar */
int last_command;               /* the last socket command */
int last_token;                 /* most recently read token for ungetToken */
int line_number;

long page_start_fpos;           /* where the current pages fpos started      */

char *read_again = 0;

char sock_buf[1024];            /* buffer for socket input */

Token token;                    /* most recently read token */
static HashTable tokenHashTable;           /* hash table of parser tokens */
StateNode *top_state_node;
Token unget_toke;

FILE *unixfd;
int useAscii; /* should we translate graphics characters on the fly */
```

```
void printPageAndFilename(void);
void printNextTenTokens(void);

extern char *token_table[];

char *token_table[] = {
  "",           /* Dummy token name */
  "word",
  "page",
  "lispcommandquit",
  "bf",
  "link",
  "downlink",
  "beginscroll",
  "spadcommand",
  "nolines",
  "env",
  "par",
  "centerline",
  "begin",
  "beginitems",
  "item",
  "table",
  "fbox",
  "tab",
  "space",
  "indent",
  "horizontalline",
  "newline",
  "enditems",
  "returnbutton",
  "memolink",
  "upbutton",
  "endscroll",
  "thispage",
  "returnto",
  "free",
  "bound",
  "lisplink",
  "unixlink",
  "mbox",
  "inputstring",
  "stringvalue",
  "spadlink",
  "inputbitmap",
```

```
"inputpixmap",
"unixcommand",
"em",
"lispcommand",
"lispmemolink",
"lispdownlink",
"spadcall",
"spadcallquit",
"spaddownlink",
"spadmemolink",
"qspadcall",
"qspadcallquit",
"inputbox",
"radioboxes",
"boxvalue",
"vspace",
"hspace",
"newcommand",
"windowid",
"beep",
"quitbutton",
"begintitems",
"titem",
"end",
"it",
"sl",
"tt",
"rm",
"ifcond",
"else",
"fi",
"newcond",
"setcond" ,
"button",
"windowlink",
"haslisp",
"hasup",
"hasreturn",
"hasreturnto",
"lastwindow",
"endtitems",
"lispwindowlink",
"beginpile",
"endpile",
"nextline",
"pastebutton",
```

```
"color",
"helppage",
"patch",
"radiobox",
"ifrecond",
"math",
"mitem",
"pagename",
"examplenumber",
"replacepage",
"inputimage",
"spadgraph",
"indentrel",
"controlbitmap"
};
```

⟨*token.h*⟩
⟨*spadErrorHandler*⟩
⟨*spadBusy*⟩
⟨*connectSpad*⟩
⟨*resetConnection*⟩
⟨*pathname*⟩
⟨*BeStruct*⟩
⟨*strpostfix*⟩
⟨*extendHT*⟩
⟨*buildHtFilename*⟩
⟨*htFileOpen*⟩
⟨*tempFileOpen*⟩
⟨*halloc*⟩
⟨*hashInit*⟩
⟨*hashInsert*⟩
⟨*hashDelete*⟩
⟨*hashMap*⟩
⟨*hashFind*⟩
⟨*hashReplace*⟩
⟨*freeHash*⟩
⟨*stringHash*⟩
⟨*stringEqual*⟩
⟨*allocString*⟩
⟨*jump*⟩
⟨*tokenName*⟩
⟨*printToken*⟩
⟨*printPageAndFilename*⟩
⟨*printNextTenTokens*⟩
⟨*parserInit*⟩
⟨*initScanner*⟩

⟨*saveScannerState*⟩
⟨*restoreScannerState*⟩
⟨*ungetChar*⟩
⟨*getExpectedToken*⟩
⟨*ungetToken*⟩
⟨*getChar1*⟩
⟨*getChar*⟩
⟨*getToken*⟩
⟨*pushBeStack*⟩
⟨*clearBeStack*⟩
⟨*checkAndPopBeStack*⟩
⟨*beType*⟩
⟨*beginType*⟩
⟨*endType*⟩
⟨*keywordType*⟩

Chapter 4

Shared include files

4.1 debug.c

⟨*debug.c*⟩≡

```
#include "debug.h"

#ifdef free
#undef free
hfree(char *p) {
  free(p);
}
#endif
```

4.2 hyper.h

The `hypertex` function, of which this is the top level, is a browser for Axiom information. It works off a database of pages. The pages are stored in the `$AXIOM/doc` subdirectory and there is a key file called `ht.db` in that subdirectory which contains critical information about each page. If you add or delete pages you must rerun the `htadd` command. (See the `htadd` command in `src/hyper/htadd.pamphlet`.)

Generally, if you add or delete pages you can recreate a proper `pages/ht.db` file by doing:

```
cd $AXIOM/doc
htadd -f pages -n pages/*
```

The `hypertex` function looks in `$AXIOM/doc` by default. This can be over-ridden by setting the `HTPATH` shell variable to point to the desired directory containing the pages and the ht.db file.

⟨*hyper.h*⟩≡

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>

#include "com.h"
⟨token.h⟩
#include "hash.h"

#define boolean unsigned short int

#ifndef TRUE
#define TRUE   ((boolean) 0x1)
#endif
#ifndef FALSE
#define FALSE  ((boolean) 0x0)
#endif

/* Struct forward declarations */

struct text_node;
struct input_box;
struct input_window;
```

```
struct paste_node;
struct radio_boxes;
struct group_item;

#define Scrollupbutton 1
#define Scrolldownbutton 2
#define Noopbutton 6

#define Scrolling 1
#define Header 2
#define Footer 3
#define Title 4

extern int MenuServerOpened;

/* These are all the colors one can use in HyperDoc. */

extern int gActiveColor;
extern int gAxiomColor;
extern int gBackgroundColor;
extern int gBfColor;
extern int gControlBackgroundColor;
extern int gControlForegroundColor;
extern int gEmColor;
extern int gInputBackgroundColor;
extern int gInputForegroundColor;
extern int gItColor;
extern int gRmColor;
extern int gSlColor;
extern int gTtColor;

/* These are all the different fonts one can use in HyperDoc. */

extern XFontStruct *gActiveFont;
extern XFontStruct *gAxiomFont;
extern XFontStruct *gBfFont;
extern XFontStruct *gEmFont;
extern XFontStruct *gInputFont;
extern XFontStruct *gItFont;
extern XFontStruct *gRmFont;
extern XFontStruct *gSlFont;
extern XFontStruct *gTitleFont;
extern XFontStruct *gTtFont;

/** I am implementing a state node stack, this is the structure I store **/
```

```
typedef struct state_node {
 int last_ch, last_token, inputType;
 long fpos, keyword_fpos;
 long page_start_fpos;
 Token token;
 char *inputString;
 FILE *cfile;
 int keyword;
 struct state_node *next;
} StateNode;

/** pointer to the top of the state node graph **/
extern StateNode *top_state_node;


/* structure for a hyper text link */
typedef struct hyper_link {
  int type;                     /* Memolink, Spadlink, Downlink etc. */
  Window win;                   /* X11 window containing active area */
  union {
    struct text_node *node;     /* ID of link to which link refers */
    struct input_box *box;
    struct input_window *string;
    struct paste_node *paste;   /* the paste node area */
  } reference;
  int x,y;                      /* relative position inside page */
} HyperLink;


typedef struct if_node {
    struct text_node *cond;     /* the condition nodes*/
    struct text_node *thennode;
    struct text_node *elsenode;
} IfNode;

typedef struct item_stack {
    int indent;
    int item_indent;
    int in_item;
    struct item_stack *next;
} ItemStack;

typedef struct paste_node {
   char *name;
   int where;              /* where should I be parsing from? */
   short int hasbutton;
```

```
    short int haspaste;
    struct group_item *group;
    ItemStack *item_stack;
    struct text_node *arg_node;
    struct text_node *end_node;
    struct text_node *begin_node;
    struct input_window *paste_item;
} PasteNode;

/* Structure for formatted hypertext */

typedef struct text_node {
  short type;                   /* type of node (text, link, etc.) */
  int x,y, width, height;       /* relative location on screen */
  int space;                    /* was there space in front of me ? */
  union {
    char *text;                 /* piece of text to display */
    struct text_node *node;     /* argument text */
    struct if_node *ifnode;
  } data;
  HyperLink *link;              /* link for active text */
  union {
    Pixmap pm;                  /* pixmap for bit images */
    XImage *xi;                 /* pixmap image */
  } image;
  struct text_node *next;       /* next node in formatted text */
} TextNode;

/** Structure used to store pixmaps and bitmaps **/

typedef struct image_struct {
   int width,height;   /** It's width and height **/
   union {
     Pixmap pm;
     XImage *xi;
   } image;
   char *filename;      /** The filename used to reference it **/
} ImageStruct;

/* Structure for locating HyperDoc pages in a source file */

typedef struct {
  char *name;           /* file name */
  long  pos;            /* position in file */
  int   ln;             /* the line number */
} FilePosition;
```

```
/*** The structure needed for storing a macro **/

typedef struct macro_store {
  short int loaded;
  FilePosition fpos;
  char *name;
  char *macro_string;
  short number_parameters;
} MacroStore;


/** Structure needed for storing a patch **/
typedef struct patch_store {
  short int loaded;
  FilePosition fpos;
  char *name;
  char *string;
} PatchStore;

/*  Here are the structures needed for doing input to HyperDoc windows. */

typedef struct line_struct {
   char *buffer;
   int changed;     /* Has the line changed */
   int line_number;
   int buff_pntr;
   int len;
   struct line_struct *prev, *next;
} LineStruct;

typedef struct input_window {
  char *name;               /* symbol name */
  int size;                 /* the length of the window */
  int cursor_x;             /* x-coordinate for the cursor **/
  int entered;              /* tells me whether I have typed here before */
  int num_lines;            /* number of lines needed to store buffer */
  LineStruct *lines;
  LineStruct *curr_line;    /* the current line on which the cursor */
  Window win;
  struct input_window  *next;
}  InputItem;


/* structure for storing input boxes **/
typedef struct input_box {
```

```
    char *name;
    ImageStruct *selected, *unselected;
    short int picked;
    struct input_box  *next;
    struct radio_boxes *rbs;
    Window win;
} InputBox;

typedef struct radio_boxes {
     char *name;
     InputBox *boxes;
     ImageStruct *selected, *unselected;
     int width, height;
     struct radio_boxes *next;
} RadioBoxes;

/* Structure for spadcommand dependencies hash table entries */
typedef struct spadcom_depend {
  char *label;                  /* dependency label */
  TextNode *spadcom;            /* spadcommand defining the label */
  short executed;               /* true iff spadcommand has benn executed */
} SpadcomDepend;

typedef struct button_list {
  int x0,y0,x1,y1;
  HyperLink *link;
  Window win;
  struct button_list *next;
} ButtonList;

/* Stucture for unformatted hyper text page */

typedef struct hyperdoc_page {
  short type;                /* Normal, Quitbutton, Upbutton etc.         */
  char *name;                /* ID of page                                */
  char *filename;            /* The name of the file for the page, or null*/
  int scroll_off;            /* The offset in the scrolling region        */
  int bot_scroll_margin;     /* bottom of the scrolling region            */
  int top_scroll_margin;     /* top of the scrolling region               */
  TextNode *title;           /* the title of the page                     */
  TextNode *header;          /* formatted version of page                 */
  TextNode *scrolling;       /* Top of scrolling region                   */
  TextNode *footer;          /* top of non-scrolling region at bottom     */
  Sock *sock;                /* socket connection for spad buffer         */
  HashTable *fLinkHashTable; /* active link hash table                    */
  ButtonList *s_button_list; /* active buttons on page                    */
```

```
  ButtonList *button_list;   /* active buttons on page                    */
  HashTable *depend_hash;    /* Hash tables of spadcommand dependencies   */
  InputItem *input_list;     /* List of input structures                  */
  InputItem *currentItem;    /* a pntr to the currently active item       */
  HashTable *box_hash;       /* place where all the boxes are stored      */
  RadioBoxes *radio_boxes;   /* a linked list of radio boxes              */
  short pageFlags;           /* A list of flags for the page              */
  char *helppage;            /* the name of the helppage                  */
} HyperDocPage;

/* Structure for an unloaded page */

typedef struct unloaded_page {
  short type;        /* indicator of unloaded page */
  char *name;        /* name of page */
  FilePosition fpos; /* where to find the page */
} UnloadedPage;

/* Structure for a HyperDoc Window */

typedef struct {
  Window fMainWindow;       /* The main text field window.          */
  Window fScrollWindow;     /* The scrolling area of the window     */
  Window fDisplayedWindow;  /* The current window of the above two, */
                            /*   being filled by display            */
  Window fScrollUpWindow;   /* Window for scrolling up a line       */
  Window fScrollDownWindow; /* Window for scrolling down a line     */
  Window scrollbar;         /* the window for scrolling             */
  Window scroller;          /* the scroller window                  */
  Window fTitleBarButton1;  /* 1st titlebar bitmap button           */
  Window fTitleBarButton2;  /* 2nd titlebar bitmap button           */
  Window fTitleBarButton3;  /* 3rd titlebar bitmap button           */
  Window fTitleBarButton4;  /* 4th titlebar bitmap button           */
  int fScrollerTopPos;      /* where the top of the scroller is     */
  int fScrollerHeight;      /* the height of the scroller           */
  int fScrollBarHeight;     /* the height for the scrollbar         */
  int scrollwidth;          /* the width of the scrolling area      */
  int scrollheight;         /* the height of the scrolling area     */
  int scrollupy;            /* Current y position of scroll up button */
  int scrolldowny;          /* Current y position of scroll down button */
  int scrollbary;           /* Current y position of teh scrollbar      */
  int scrollx;              /* X coordinates for all of the above       */
  int border_width;         /* Width of the border                      */
  HyperDocPage *page;       /* currently displayed page                 */
  int width;                /* in pixels                                */
  int height;               /* in pixels                                */
```

```
  int columns;              /* Width in chars, only setable for form pages */
  HyperDocPage **fMemoStack; /* stack of memo links */
  HyperDocPage **fDownLinkStack;/* stack of down links */
  int *fDownLinkStackTop;       /* stack of down links */
  int fMemoStackIndex;          /* memo stack pointer */
  int fDownLinkStackIndex;      /* downlink stack pointer */
  HashTable *fWindowHashTable;  /* hash table of active subwindows */
  HashTable *fPageHashTable;    /* hash table of HyperDoc pages */
  HashTable *fPasteHashTable;   /* hash table for paste in areas */
  HashTable *fMacroHashTable;   /* hash table of HyperDoc macros */
  HashTable *fCondHashTable;    /* hash table for values         */
  HashTable *fPatchHashTable;   /* hash table for patch locations */
  int fAxiomFrame;              /* Axiom frame number initializing window */
  GC fStandardGC;               /* Graphics context for window */
  GC fInputGC;                  /* Graphics context for the input windows */
  GC fCursorGC;                 /* Graphics context for the cursors */
  GC fControlGC;                /* Graphics context for the buttons */
  Cursor fDisplayedCursor;      /* The currently displayed cursor */
} HDWindow;

/* Structure for identifying appropriate link hash tables */

typedef struct {
  int code;           /* code of active area */
  HyperDocPage *page; /* page for which hash table applies */
} LinkHashID;

/*** Flags for the page ***/

#define NOLINES 0000001 /* Ibid, for the bottom of the page */


/* external variables and functions.  See the source file for a description
 of their purposes */

extern HashTable gSessionHashTable;   /* hash table of HD windows */

extern HDWindow *gParentWindow;       /* the parent window. The one that
                                       * appears when you first start HD */

extern HyperLink *quitLink; /** a special link to the protected quit page **/


/* From hyper.c */
extern int gXScreenNumber;
extern Display *gXDisplay;
```

```
extern int gSwitch_to_mono;
extern unsigned long * spadColors;
extern int gIsEndOfOutput;
extern HDWindow *gWindow;
extern Sock *sessionServer;
extern Sock *spadSocket;
extern HashTable gFileHashTable;
extern HashTable gImageHashTable;    /* A global hash table for images */
extern Cursor gNormalCursor;         /* The normal mouse cursor */
extern Cursor gActiveCursor;         /* The cursor in active regions */
extern Cursor gBusyCursor;           /* The clock cursor for when I am busy */
extern int gIsAxiomServer;  /* true iff HyperDoc is acting as an Axiom server*/
extern int gArgc;                    /* original argc from main */
extern char **gArgv;                 /* original argv from main */
/* from lex.c */
extern long fpos, keyword_fpos;
extern Token token;
extern int last_token, inputType, last_ch;
extern char *inputString;
extern FILE *cfile;
/* from input.c */
extern XImage *picked;
extern int picked_height;
extern int picked_width;
extern XImage *unpicked;
extern int unpicked_height;
extern int unpicked_width;
/* from display.c */
extern int line_height;
extern int need_scroll_up_button;
extern int scrolling;
extern int need_scroll_down_button;
extern int space_width;

#define NoChar -9999
#define temp_dir "/tmp/"
#define dbFileName "ht.db"
#define def_spad "/usr/local/axiom"


/* Types of HyperDoc pages */

#define UlUnknownPage    9993 /*I hate this hack, but I have to know whether*/
#define UnknownPage      9994 /*this page has been loaded or not. */
#define ErrorPage        9995
#define Unixfd           9996
```

```
#define SpadGen          9997
#define Normal           9998
#define UnloadedPageType 9999

/* Commands from Axiom */

#define EndOfPage        99
#define SendLine         98
#define StartPage        97 /* A normal HyperDoc page */
#define LinkToPage       96
#define PopUpPage        95 /* A pop-up page*/
#define PopUpNamedPage   94
#define KillPage         93
#define ReplacePage      92
#define ReplaceNamedPage 91
#define SpadError        90

/* Constants declaring size of page stacks */

#define MaxMemoDepth 25 /* max nesting level for memolinks */
#define MaxDownlinkDepth 50 /* max downlink nesting depth */

/* Constants defining the size of various hash tables */

#define PageHashSize     1000
#define FileHashSize     30
#define SessionHashSize  10
#define MacroHashSize    100
#define ImageHashSize    100
#define CondHashSize     100
#define BoxHashSize      20
#define PasteHashSize    100
#define PatchHashSize    100

/* A couple of macros for memo and down links */

#define need_up_button \
  (gWindow->fMemoStackIndex ? gWindow->fDownLinkStackIndex >= \
   gWindow->fDownLinkStackTop[gWindow->fMemoStackIndex-1] \
   : gWindow->fDownLinkStackIndex)

#define need_return_button (gWindow->fMemoStackIndex)

#define need_help_button (gWindow->page->helppage != NULL)

#define max(x,y) ((x) > (y) ? (x) : (y))
```

```
#define pick_box(box) fillBox(box->win, box->selected)
#define unpick_box(box) fillBox(box->win, box->unselected)

#define TopLevelHelpPage  "ugHyperPage"
#define NoMoreHelpPage    "NoMoreHelpPage"
#define KeyDefsHelpPage   "ugHyperKeysPage"
#define InputAreaHelpPage "ugHyperInputPage"

/* definitions for connecting to the Axiom server */

#define Connected 0
#define NotConnected 1
#define SpadBusy 2

/* some GUI-dependent stuff */

#define BeepAtTheUser()     /* (XBell(gXDisplay,  5)) */
#define LoudBeepAtTheUser() /* (XBell(gXDisplay, 50)) */


#if defined(RTplatform) || defined(PS2platform) || defined(RIOSplatform) || defin
#define RmFontDefault         "Rom14"
#define TtFontDefault         "Erg14"
#define ActiveFontDefault     "Bld14"
#define AxiomFontDefault      "Erg14"
#define EmphasizeFontDefault  "Itl14"
#define BoldFontDefault       "Bld14"
#endif

#if defined(SUNplatform) || defined (SUN4OS5platform) || defined(SGIplatform) || 
#define RmFontDefault "-adobe-courier-medium-r-normal--18-*-*-*-m-*-iso8859-1"
#define TttFontDefault "-adobe-courier-medium-r-normal--18-*-*-*-m-*-iso8859-1"
#define ActiveFontDefault "-adobe-courier-bold-r-normal--18-*-*-*-m-*-iso8859-1"
#define AxiomFontDefault "-adobe-courier-bold-o-normal--18-*-*-*-m-*-iso8859-1"
#define EmphasizeFontDefault "-adobe-courier-medium-o-normal--18-*-*-*-m-*-iso885
#define BoldFontDefault "-adobe-courier-bold-r-normal--18-*-*-*-m-*-iso8859-1"
#endif




typedef struct group_item {
```

```
    int cur_color;
    XFontStruct *cur_font;
    int center;
    struct group_item *next;
} GroupItem;


extern GroupItem   *gTopOfGroupStack;


typedef struct cond_node {
   char *label;
   char *cond;
} CondNode;

typedef struct parameter_list_type {
    char          **list;       /** The parameters in string form **/
    short           number;     /** How many parameters are there **/
    struct parameter_list_type *next;
}              *ParameterList;
```

Chapter 5

The spadbuf function

5.1 Constants and Headers

5.1.1 System includes

⟨*spadbuf*⟩≡

```
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/time.h>
#include <signal.h>
```

5.1.2 Local includes

⟨*spadbuf*⟩+≡

```
#include "debug.h"
#include "bsdsignal.h"
#include "edible.h"
#include "com.h"
#include "spadbuf.h1"
#include "bsdsignal.h1"
#include "sockio-c.h1"
#include "edin.h1"
#include "wct.h1"
#include "prt.h1"
#include "cursor.h1"
#include "fnct-key.h1"
```

5.2 externs

⟨*spadbuf*⟩+≡

```
extern char in_buff[];        /* buffer for storing characters read
                                 until they are processed */
extern char buff[];         /* Buffers for collecting input and */
extern int buff_flag[];    /* flags for whether buff chars
                                 are printing or non-printing */
```

5.3 local variables

⟨*spadbuf*⟩+≡
```
unsigned char _INTR, _QUIT, _ERASE, _KILL, _EOF, _EOL, _RES1, _RES2;
int contNum;                    /* do reading and all the other fun stuff
                                 * depend on this for all there ioctl's */
int num_read;

/*
 * Here are the term structures I need for setting and resetting the terminal
 * characteristics.
 */
struct termios oldbuf;     /* the initial settings */
struct termios canonbuf;   /* set it to be canonical */
struct termios childbuf;

short INS_MODE;            /* Flag for insert mode */
short ECHOIT;              /* Flag for echoing */
short PTY;
int MODE;                  /* Am I in cbreak, raw, or canonical */

char in_buff[1024];        /* buffer for storing characters read
                              until they are processed */
char buff[MAXLINE];        /* Buffers for collecting input and */
int  buff_flag[MAXLINE];   /* flags for whether buff chars
                              are printing or non-printing */
int (*old_handler) ();
Sock *session_sock, *menu_sock;
char *buff_name = NULL;    /* name for the aixterm */
```

5.4 Code

This routine used to be used to send sigint onto spad, but now they go through just fine on their own reinstated for AIX V3.2

5.4.1 spadbufInterHandler

⟨*spadbuf*⟩+≡
```
static void spadbufInterHandler(int sig) {
    send_signal(session_sock, SIGUSR2);
}
```

5.4.2 spadbufFunctionChars

⟨*spadbuf*⟩+≡

```
static void spadbufFunctionChars(void) {
    /** once I have that get the special characters        ****/
    _INTR = oldbuf.c_cc[VINTR];
    _QUIT = oldbuf.c_cc[VQUIT];
    _ERASE = oldbuf.c_cc[VERASE];
    _KILL = oldbuf.c_cc[VKILL];
    _EOF = oldbuf.c_cc[VEOF];
    _EOL = oldbuf.c_cc[VEOL];
    return;
}
```

5.4.3 interpIO

Act as terminal session for sock connected to stdin and stdout of another process.

⟨*spadbuf*⟩+≡

```
static void interpIO(void) {
    char buf[1024];
    fd_set rd;
    int len, command;
    while (1) {
        FD_ZERO(&rd);
        FD_SET(menu_sock->socket, &rd);
        FD_SET(session_sock->socket, &rd);
        FD_SET(1, &rd);
        len = sselect(FD_SETSIZE, &rd, 0, 0, NULL);
        if (len == -1) {
            perror("stdio select");
            return;
        }
        if (FD_ISSET(session_sock->socket, &rd)) {
            len = sread(session_sock, buf, 1024, "stdio");
            if (len == -1)
                return;
            else {
                write(1, buf, len);
            }
        }
        if (FD_ISSET(menu_sock->socket, &rd)) {
            command = get_int(menu_sock);
            switch (command) {
              case -1:
                exit(0);
              case ReceiveInputLine:
                get_string_buf(menu_sock, in_buff, 1024);
                num_read = strlen(in_buff);
                clear_buff();
                do_reading();
                break;
              case TestLine:
                break;
              default:
                break;
            }
        }
        if (FD_ISSET(1, &rd)) {
```

```
            num_read = read(0, in_buff, 1024);
            do_reading();
        }
    }
}
```

5.4.4

⟨*spadbuf*⟩+≡

```
static void initParent(void) {
    /** get the original termio settings, so I never have to check again **/
    if (tcgetattr(0,&oldbuf) == -1) {
        perror("Clef Trying to get terms initial settings");
        exit(-1);
    }
    /** get the settings for my different modes ***/
    if (tcgetattr(0,&canonbuf) == -1) {
        perror("Clef Getting terminal settings");
        exit(-1);
    }
    /*** set the buffer to read before an eoln is typed ***/
    canonbuf.c_lflag &= ~(ICANON | ECHO | ISIG);
    canonbuf.c_lflag |= ISIG;

    /***  Accordingly tell it we want every character ***/
    canonbuf.c_cc[VMIN] = 1;          /* we want every character  */
    canonbuf.c_cc[VTIME] = 1;         /* these may require tweaking */

    if (tcsetattr(0, TCSAFLUSH, &canonbuf) == -1) {
        perror("Spadbuf setting parent to canon");
        exit(0);
    }
    /*
     * This routine is in edin.c and sets the users preferences for function
     * keys. In order to use it I have to set childbuf to be the same as
     * oldbuf
     */
    spadbufFunctionChars();
    INS_MODE = 0;
    ECHOIT = 1;
    Cursor_shape(2);
}
```

5.4.5 main

Modified on 6/13/90 for the command line completion abiltities of Since I am only calling this program from within spadint, I decided that the usage should be.

```
spadbuf page_name [completion_ files]
```

⟨*spadbuf*⟩+≡

```
int main(int argc,char **  argv) {
    FILE *fopen();
    if (argc < 2) {
        fprintf(stderr, "Usage : spadbuf page_name [completion_files] \n");
        exit(-1);
    }
    buff_name = *++argv;
    while (*++argv) {
        load_wct_file(*argv);
    }
    skim_wct();
    session_sock=connect_to_local_server(SessionServer, InterpWindow, Forever);
    menu_sock = connect_to_local_server(MenuServerName, InterpWindow, Forever);
    bsdSignal(SIGINT, spadbufInterHandler,RestartSystemCalls);
    /*
     * set contNum so it is pointing down the socket to the childs
     */
    contNum = session_sock->socket;
    send_string(menu_sock, buff_name);
    initParent();
    define_function_keys();
    init_reader();
    PTY = 0;
    interpIO();
    return(1);
}
```

Chapter 6

The ex2ht function

The ex2ht command creates a cover page for structured HyperDoc example pages

6.1 Constants and Headers

6.1.1 System includes

⟨*ex2ht*⟩≡

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
```

6.1.2 Local includes

⟨*ex2ht*⟩+≡

```
#include "debug.h"
#include "ex2ht.h1"
```

6.2 defines

⟨*ex2ht*⟩+≡
```
#define MaxLineLength 512
#define MaxFiles      100
```

6.3 local variables

⟨*ex2ht*⟩+≡
```
char *files[MaxFiles];
int numFiles = 0;
struct timeval latest_date[2] ={{0,0},{0,0}};
FILE *coverFile;
```

6.4 Code

6.4.1 allocString

⟨*ex2ht*⟩+≡
```
char *allocString(char *s) {
    char *t = (char *) malloc(strlen(s) + 1);
    strcpy(t, s);
    return t;
}
```

6.4.2 strPrefix

⟨*ex2ht*⟩+≡
```
char *strPrefix(char *prefix, char *s) {
    while (*prefix != '\0' && *prefix == *s) {
        prefix++;
        s++;
    }
    if (*prefix == '\0')
        return s;
    return NULL;
}
```

6.4.3 getExTitle

⟨*ex2ht*⟩+≡

```
char *getExTitle(FILE *inFile, char *line) {
    char *title;
    while (fgets(line, MaxLineLength, inFile) != NULL)
        if ((title = strPrefix("% Title: ", line))) {
            title[strlen(title) - 1] = '\0';
            return title;
        }
    fprintf(stderr, "No Title title line in the file!\n");
    return NULL;
}
```

6.4.4 exToHt

⟨*ex2ht*⟩+≡

```
void exToHt(char *filename) {
    char line[MaxLineLength], *line2;
    char *title, *pagename;
    FILE *inFile = fopen(filename, "r");
    FILE *outFile;
    int len, i;
    struct timeval  tvp;
    struct stat buf;
    if (inFile == NULL) {
        fprintf(stderr, "couldn't open %s for reading.\n", filename);
        return;
    }
    strcpy(line, "Menu");
    strcat(line, filename);
    len = strlen(line);
    for (i = 0; i < len; i++)
        if (line[i] == '.') {
            line[i] = '\0';
            break;
        }
    outFile = fopen(line, "w");
    if (outFile == NULL) {
        fprintf(stderr, "couldn't open %s for writing.\n", line);
        return;
    }
    pagename = allocString(line);
    title = getExTitle(inFile, line);
    if (title == NULL) {
        return;
    }
    files[numFiles++] = pagename;
    emitCoverLink(pagename, title);
    emitHeader(outFile, pagename, title);
    while (fgets(line, MaxLineLength, inFile) != NULL) {
        if ((line2 = strPrefix("\\begin{page}{", line)))
            emitMenuEntry(line2, outFile);
        else if ((line2 = strPrefix("\\spadcommand{", line)))
            emitSpadCommand(line2, "\\spadcommand{", outFile);
        else if ((line2 = strPrefix("\\spadpaste{", line)))
            emitSpadCommand(line2, "\\spadpaste{", outFile);
        else if ((line2 = strPrefix("\\example{", line)))
            emitSpadCommand(line2, "\\example{", outFile);
        else if ((line2 = strPrefix("\\graphpaste{", line)))
```

```
            emitSpadCommand(line2, "\\graphpaste{", outFile);
    }
    emitFooter(outFile);
    fclose(inFile);
    fclose(outFile);
    stat(filename,&buf);
    tvp.tv_sec =buf.st_mtime;
    tvp.tv_usec =0;
    if timercmp(&tvp,&latest_date[1],>){
      latest_date[1].tv_sec=buf.st_mtime;
    }
}
```

6.4.5 emitHeader

⟨*ex2ht*⟩+≡

```
void emitHeader(FILE *outFile, char *pageName, char *pageTitle) {
    fprintf(outFile, "\\begin{page}{%s}{%s}\n", pageName, pageTitle);
    fprintf(outFile, "\\beginscroll\\beginmenu\n");
}
```

6.4.6 emitFooter

⟨*ex2ht*⟩+≡

```
void emitFooter(FILE *outFile) {
    fprintf(outFile, "\\endmenu\\endscroll\\end{page}\n");
}
```

6.4.7 emitMenuEntry

s is "pageName}{title}"

⟨*ex2ht*⟩+≡

```
void emitMenuEntry(char *line, FILE *outFile) {
    char pageName[MaxLineLength], title[MaxLineLength];
    char *p = pageName, *t = title;
    while (*line != '}')
        *p++ = *line++;
    *p = '\0';
    line++;
    while (*line != '}')
        *t++ = *line++;
    *t = '\0';
    fprintf(outFile, "\\menudownlink%s}{%s}\n", title, pageName);
}
```

6.4.8 emitSpadCommand

⟨*ex2ht*⟩+≡

```
void emitSpadCommand(char *line, char *prefix, FILE *outFile) {
    int braceCount = 1;
    char command[MaxLineLength], *t = command;
    while (1) {
        if (*line == '}')
            braceCount--;
        if (braceCount == 0)
            break;
        if (*line == '{')
            braceCount++;
        *t++ = *line++;
    }
    *t = '\0';
    fprintf(outFile, "%s%s}\n", prefix, command);
}
```

6.4.9 openCoverPage

⟨*ex2ht*⟩+≡
```
void openCoverPage(void) {
    coverFile = fopen("coverex.ht", "w");
    if (coverFile == NULL) {
        fprintf(stderr, "couldn't open coverex.ht for writing\n");
        exit(-1);
    }
    fprintf(coverFile, "%% DO NOT EDIT! Created by ex2ht.\n\n");
    fprintf(coverFile, "\\begin{page}{ExampleCoverPage}{Examples Of AXIOM Commands}\n");
    fprintf(coverFile, "\\beginscroll\\table{\n");
}
```

6.4.10 closeCoverPage

⟨*ex2ht*⟩+≡
```
void closeCoverPage(void) {
    fprintf(coverFile, "}\\endscroll\\end{page}\n\n");
}
```

6.4.11 closeCoverFile

⟨*ex2ht*⟩+≡
```
void closeCoverFile(void) {
    fclose(coverFile);
    utimes("coverex.ht",latest_date);
}
```

6.4.12 emitCoverLink

⟨*ex2ht*⟩+≡
```
void emitCoverLink(char *name, char *title) {
    fprintf(coverFile, "{\\downlink{%s}{%s}}\n", title, name);
}
```

6.4.13 addFile

⟨*ex2ht*⟩+≡

```
void addFile(char *filename) {
    FILE *file = fopen(filename, "r");
    int c;

    if (file == NULL) {
        fprintf(stderr, "Couln't open %s for reading\n", filename);
        exit(-1);
    }
    while ((c = getc(file)) != EOF)
        putc(c, coverFile);
    putc('\n', coverFile);
    fclose(file);
    unlink(filename);
}
```

6.4.14 main

⟨*ex2ht*⟩+≡

```
int main(int argc, char **argv){
    int i;
    if (argc == 1) {
        fprintf(stderr, "usage: %s exfile.ht ...\n", argv[0]);
        return (-1);
    }
    openCoverPage();
    for (i = 1; i < argc; i++)
        exToHt(argv[i]);
    closeCoverPage();
    for (i = 0; i < numFiles; i++)
        addFile(files[i]);
    closeCoverFile();
    return 0;
}
```

Chapter 7

The htadd command

The `htadd` function can manipulate the database of hypertex pages. To rebuild the hypertex database changes to the `$AXIOM/doc` subdirectory and type:

```
htadd -f pages -n pages/*
```

This will create a file called `pages/ht.db` which contains entries similar to:

```
        algebra.ht 1102052108
\page AlgebraPage 216 9
\page NumberTheoryPage 763 28
        ALIST.ht 1102052108
\newcommand AssociationListXmpTitle 140 3
\newcommand AssociationListXmpNumber 195 4
\page AssociationListXmpPage 313 7
        ALIST.pht 1102052108
\patch AssociationListXmpPagePatch1 0 1
\patch AssociationListXmpPageEmpty1 447 11
...
```

7.1 Constants and Headers

7.1.1 System includes

⟨*htadd*⟩≡

```
#include <sys/stat.h>
#include <errno.h>
#include <setjmp.h>
#include <ctype.h>
```

7.1.2 structs

⟨*htadd*⟩+≡

```
typedef struct toke {   /* HyperDoc parser tokens */
  int type;             /* token type. One of those listed below */
  char *id;             /* string value if type == Identifier */
} Token;
```

7.1.3 Local includes

⟨*htadd*⟩+≡

```
⟨hyper.h⟩
#include "htadd.h1"
#include "addfile.h1"
#include "halloc.h1"
#include "hash.h1"
#include "hterror.h1"
#include "lex.h1"
```

7.1.4 extern references

⟨*htadd*⟩+≡

```
extern HyperDocPage *gPageBeingParsed;
extern short int gInSpadsrc;
extern short int gInVerbatim;
extern int line_number;         /* keeps track of which line a page starts on
                                 * in a file. This way someone can start
                                 * including a line number counter into
                                 * HyperDoc. */
```

7.1.5 defines

⟨*htadd*⟩+≡

```
#define Delete 1
#define System 2
#define Current 4
#define Named 8
#define ptype(c, t) (strcpy(c, t));
#define Special(t) (( t == Page || t == NewCommand || t == Patch )?(1):(0))
#define usage "usage: htadd [-s|-l|-f db-directory] [-d|-n] filenames"
#define special(c) ((c) == '{' || (c) == '}' || (c) == '#' || (c) == '%' || \
                    (c) == '\\'  || (c) == '[' || (c) == ']' || (c) == '_' || \
                    (c) == ' ' || (c) == '$' || (c) == '~' || (c) == '^' ||  \
                    (c) == '&')

#define punctuation(c) ((c)== '`' || (c) == '\'' || (c) == ','  || \
                        (c) == '.' || (c) == '?' || (c) == '"' || \
                        (c)  == ';' || (c) == ':' || (c) == '-')

#define whitespace(c) ((c) == ' ' || (c) == '\t' || (c) == '\n')
#define delim(c) \
  (whitespace(c) || special(c) || punctuation(c))
#define filedelim(c) \
  (whitespace(c))
```

7.1.6 forward declarations

⟨*htadd*⟩+≡

```
static void updateDB(FILE *db, FILE *temp_db, FILE *new_file,
          char *addname, char *fullname, int fresh);
static void addNewPages(FILE *temp_db, FILE *new_file,
                        char *addname, char *fullname);
static void copyFile(char *f1, char *f2);
static void getFilename(void);
static void deleteDB(FILE *db, FILE *temp_db, char *name);
FILE *htFileOpen(char *fname, char *aname, char *name);
FILE *tempFileOpen(char *temp_dbFile);
char *allocString(char *str);
void printNextTenTokens(void);
int getToken(void);
int keywordType(void);
```

7.1.7 local variables

⟨*htadd*⟩+≡

```
int fresh = 0;

int MenuServerOpened;

int gTtFontIs850=0;
HDWindow *gWindow = NULL;
Display *gXDisplay;
int gXScreenNumber;

Sock *sessionServer = NULL;
Sock *spadSocket = NULL;
int still_reading;
int str_len;
```

7.2 The Shared Code

⟨*htadd*⟩+≡

⟨*htadd shared code*⟩

7.3 Code

7.3.1 parseArgs

This routine parses the command line arguments. It parses the command line arguments. It returns a flag which tells the calling routine what database file to use, and whether or not to delete files.

⟨*htadd*⟩+≡

```
static void parseArgs(char **argv, char *db_dir, char **filenames, short *fl) {
    *fl = 0;
    while (*++argv) {
        if (!strcmp(*argv, "-d"))
            *fl |= Delete;
        else if (!strcmp(*argv, "-s")) {
            if (*fl & Current || *fl & Named) {
                fprintf(stderr, "%s\n", usage);
                exit(-1);
            }
            *fl |= System;
        }
        else if (!strcmp(*argv, "-n")) {
            fresh = 1;
        }
        else if (!strcmp(*argv, "-l")) {
            if (*fl & System || *fl & Named) {
                fprintf(stderr, "%s\n", usage);
                exit(-1);
            }
            *fl |= Current;
        }
        else if (!strcmp(*argv, "-f")) {
            if (*fl & System || *fl & Current) {
                fprintf(stderr, "%s\n", usage);
                exit(-1);
            }
            *fl |= Named;
            strcpy(db_dir, *++argv);
        }
        else
            *filenames++ = *argv;
    }
    *filenames = NULL;
}
```

7.3.2 writable

Check to see if the user has permission

⟨*htadd*⟩+≡

```
static int writable(struct stat buff) {
#ifdef DEBUG
    unsigned short uid = geteuid(), gid = getegid();
    fprintf(stderr, "Uid = %d and Gid = %d\n", uid, gid);
#endif
    /*
     * Checks the status structure sent against the user id, and goup id
     */
    if ((buff.st_uid == geteuid()) && (buff.st_mode & S_IWUSR))
        return 1;
    else if ((buff.st_gid == getegid()) && (buff.st_mode & S_IWGRP))
        return 1;
    else if ((buff.st_mode & S_IWOTH))
        return 1;
    return 0;
}
```

7.3.3 buildDBFilename

This procedure builds the db filename. Subsequently, it is passed onto all the add files that are called.

⟨*htadd*⟩+≡

```
static int buildDBFilename(short flag, char *db_dir, char *dbfilename) {
    int ret_status;
    struct stat buff;
    char *SPAD;
    char path[256];
    if (flag & System) {
        SPAD = (char *) getenv("AXIOM");
        if (SPAD == NULL) {
            fprintf(stderr,
                    "buildDBFilename: Defaulting on $AXIOM\n");
            SPAD = (char *) def_spad;
        }
        sprintf(dbfilename, "%s/doc/%s", SPAD, dbFileName);
        sprintf(path, "%s/doc", SPAD);
    }
    else if (flag & Named) {
        sprintf(dbfilename, "%s/%s", db_dir, dbFileName);
        strcpy(path, db_dir);
    }
    else {                      /* use the current directory */
        sprintf(dbfilename, "./%s", dbFileName);
        sprintf(path, "./");
    }
/*    fprintf(stderr,"htadd:buildDBFilename:dbfilename=%s\n",dbfilename);*/
    /* Now see if I can write to the file  */
    ret_status = stat(dbfilename, &buff);
    if (ret_status == -1) {
        if (errno == ENOENT) {
            /* If the file does not exist, then check it's path */
            ret_status = stat(path, &buff);
        }
        if (ret_status == -1) {
            perror("build_dbFile");
            exit(-1);
        }
    }
    /* check the status */
    if (writable(buff))
        return 1;
    fprintf(stderr, "buildDBFilename: Database file name is not writable\n");
```

```
    exit(-1);
    return 0;
}
```

7.3.4 addfile

This procedure now works as follows:

1. It adds the files to the dbFile without full pathnames.
   Two names are going to be used when adding a file -
   - addname ¡– The name without any paths
   - fullname ¡– The name with a path prepended to it
2. If the user specifies a pathname, then it is the path name that is used. If the user does not specify a path name, then possible paths are found as follows:
   - If the user has an environment variable HTPATH set, the paths mentioned are used.
   - If not, then the $AXIOM environment variable is used.

⟨*htadd*⟩+≡

```
static void addfile(char *dbname, char *name, int fresh) {
    char fullname[256];
    char temp_dbFile[256];
    FILE *db_fp = NULL;
    FILE *temp_db_fp = NULL;
    FILE *ht_fp = NULL;
    char addname[100];
    /*char *HTPATH;*/
    /*char *trace;*/
    /*char *spad;*/
    /** First thing I should do is find the proper file and open it **/
    ht_fp = htFileOpen(fullname, addname, name);
    /*
     * Now I should try to open the two database files. The one to work with,
     * and the temporary one; Send it a 1 so it checks for write access
     */
    if (fresh) {
        if ((db_fp = fopen(dbname, "a")) == NULL) {
            fprintf(stderr, "Can't open database: %s file for appending\n",
                    dbname);
            exit(-1);
        }
    }
    else {
        if ((db_fp = fopen(dbname, "r")) == NULL) {
        }
    }
```

```
    if (!fresh)
        temp_db_fp = tempFileOpen(temp_dbFile);
    /** Now actually update the file by adding the changes ***/
    updateDB(db_fp, temp_db_fp, ht_fp, addname, fullname, fresh);
    if (!fresh)
        fclose(temp_db_fp);
    fclose(ht_fp);
    if (db_fp != NULL)
        fclose(db_fp);
    if (!fresh) {
        copyFile(temp_dbFile, dbname);
        unlink(temp_dbFile);
    }
}
```

7.3.5 updateDB

⟨*htadd*⟩+≡

```
static void updateDB(FILE *db, FILE *temp_db, FILE *new_file,
          char *addname, char *fullname, int fresh) {
    /*fprintf(stderr,"TPDHERE:updateDB:addname=%s fullname=%s fresh=%d/n",
            addname,fullname,fresh); */
    char *fname;
    int c, file_there = 0, mtime;
    if (fresh) {
        addNewPages(db, new_file, addname, fullname);
        return;
    }
    if (db == NULL) {
        addNewPages(temp_db, new_file, addname, fullname);
        return;
    }
    initScanner();
    cfile = db;
    c = getChar();
    do {
        if (c == '\t') {
            getFilename();
            fname = allocString(token.id);
            getToken();
            mtime = atoi(token.id);
            if (strcmp(fname, addname) == 0) {
                saveScannerState();
                addNewPages(temp_db, new_file, addname, fullname);
                restoreScannerState();
                file_there = 1;
                while ((c = getChar()) != EOF) {
                    if (c == '\t')
                        break;
                }
            }
            else {
                fprintf(temp_db, "\t%s %d", fname, mtime);
                while ((c = getChar()) != EOF) {
                    if (c == '\t')
                        break;
                    putc(c, temp_db);
                }
            }
            free(fname);
        }
```

```
        else
            c = getChar();
    } while (c != EOF);
    if (!file_there) {
        addNewPages(temp_db, new_file, addname, fullname);
    }
}
```

7.3.6 addNewPages

⟨*htadd*⟩+≡

```
static void addNewPages(FILE *temp_db, FILE *new_file,
                        char *addname, char *fullname) {
    char type[15];
    int pos;
    int present_type;
    int pages = 0;
    struct stat fstats;
    stat(fullname, &fstats);
    fprintf(temp_db, "\t%s %d\n", addname, (int)fstats.st_mtime);
    cfile = new_file;
    initScanner();
    while (getToken() != EOF) {
        if (Special(token.type)) {
            ptype(type, token.id);
            present_type = token.type;
            pos = keyword_fpos;
            getToken();
            if (token.type != Lbrace) {
                fprintf(stderr,"missing left brace after a page, macro ");
                fprintf(stderr,"or patch declaration\n In the file ");
                fprintf(stderr,"%s on line %d\n", fullname, line_number);
                exit(-1);
            }
            getToken();
            if (present_type == Page && token.type != Word) {
                fprintf(stderr, "missing page name after \\begin{page}\n");
                fprintf(stderr,
                  "In the file %s on line %d\n", fullname, line_number);
                exit(-1);
            }
            else if (present_type == Macro && token.type != Macro) {
                fprintf(stderr, "Expected a \\macro name after newcommand, ");
                fprintf(stderr,"got %s\n",token.id);
                fprintf(stderr, "In the file %s on line %d\n",
                        fullname, line_number);
                exit(-1);
            }
            else if (present_type == Patch && token.type != Word) {
                fprintf(stderr, "Missing patch name after a \\begin{patch}\n");
                fprintf(stderr, "In the file %s on line %d\n",
                        fullname, line_number);
                exit(-1);
            }
```

```
            fprintf(temp_db, "\\%s %s %d %d\n", type,
                    token.id, pos, line_number);
            pages++;
        }
    }
    printf("Added %3d pages and/or macros from %s\n", pages, addname);
}
```

7.3.7 copyFile

⟨*htadd*⟩+≡

```
static void copyFile(char *f1, char *f2) {
    FILE *fp1, *fp2;
    int c;
    fp1 = fopen(f1, "r");
    fp2 = fopen(f2, "w");
    while ((c = getc(fp1)) != EOF) {
        putc(c, fp2);
    }
    fclose(fp2);
    fclose(fp1);
}
```

7.3.8 getFilename

⟨*htadd*⟩+≡

```
static void getFilename(void) {
    int c, ws;
    static char buffer[256];
    char *buf = buffer;
    do {
        keyword_fpos = fpos;
        c = getChar();
        ws = whitespace(c);
    } while (ws);
    switch (c) {
      case EOF:
        fprintf(stderr, "Error trying to read ht.db, unexpected EOF\n");
        exit(-1);
      case '%':
      case '\\':
      case '{':
      case '}':
        fprintf(stderr, "Error unexpexted character %c\n",c);
        exit(-1);
      default:
        do {
            *buf++ = c;
        } while ((c = getChar()) != EOF && !filedelim(c));
        ungetChar(c);
        *buf = '\0';
        token.type = Word;
        token.id = buffer;
        break;
    }
}
```

7.3.9 deleteFile

⟨*htadd*⟩+≡

```
static int deleteFile(char *dbname, char *name) {
    char temp_dbFile[256];
    FILE *db_fp, *temp_db_fp;
    char dname[256];
    strcpy(dname, name);
    extendHT(dname);
    /* Open both the tmp database and the real one */
    if ((db_fp = fopen(dbname, "r")) == NULL) {
        fprintf(stderr, "database file is empty, nothing to delete\n");
        return 1;
    }
    temp_db_fp = tempFileOpen(temp_dbFile);
    /** Now actually update the file by deleting the pages */
    deleteDB(db_fp, temp_db_fp, dname);
    fclose(temp_db_fp);
    if (db_fp != NULL)
        fclose(db_fp);
    copyFile(temp_dbFile, dbname);
    unlink(temp_dbFile);
    return 0;
}
```

7.3.10 deleteDB

⟨*htadd*⟩+≡

```
static void deleteDB(FILE *db, FILE *temp_db, char *name) {
    char *fname;
    int c/*, file_there = 0*/, mtime;
    initScanner();
    cfile = db;
    c = getChar();
    do {
        if (c == '\t') {
            getFilename();
            fname = allocString(token.id);
            getToken();
            mtime = atoi(token.id);
            if (strcmp(fname, name) == 0) {
                while ((c = getChar()) != EOF) {
                    if (c == '\t')
                        break;
                }
            }
            else {
                fprintf(temp_db, "\t%s %d", fname, mtime);
                while ((c = getChar()) != EOF) {
                    if (c == '\t')
                        break;
                    putc(c, temp_db);
                }
            }
            free(fname);
        }
        else
            c = getChar();
    } while (c != EOF);
}
```

7.3.11 main

⟨*htadd*⟩+≡

```
int main(int argc, char **argv) {
    /*int i;*/
    char db_dir[256];           /* the directory where the db file is */
    char dbfilename[256];       /* the database filename */
    char *filenames[1000];      /* the files to be added */
    char **fnames = filenames;
    short flag;                 /* flag for deleting or adding */
    parseArgs(argv, db_dir, filenames, &flag);
    if (!filenames[0]) {
        fprintf(stderr, "%s\n", usage);
        return -1;
    }
    parserInit();
    buildDBFilename(flag, db_dir, dbfilename);
    if (fresh)
        unlink(dbfilename);
    if (flag & Delete)
        while (*fnames)
            deleteFile(dbfilename, *fnames++);
    else
        while (*fnames)
            addfile(dbfilename, *fnames++, fresh);
    return 0;
}
```

Chapter 8

The hthits function

This source file implements HyperDoc's ability to scan files for a given pattern. For that purpose it needs a "regex" for string pattern matching.

This source file used to rely on `<regexp.h>` which was originally part of the X/Open System Interface and Headers Issue 2. However, since then, it has been withdrawn and no longer always available on newer platfroms. Consequently, we need to use a different, portable regex library. The POSIX definition provides one, namely through `<regex.h>`. That is what we use now. Its availability is tested at configure time.

```
hthits pattern htdb-file
```

Scan HyperDoc files for a given pattern.

The output contains lines of the form:

```
page-name‘title‘n
```

The title and body of each page are scanned but the name is not. It is possible that the title matches but not any lines. The number of matches in the page (n) is given last. (SMW Feb 91)

8.1 Constants and Headers

8.1.1 System includes

⟨*hthits*⟩≡

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <regex.h>
```

8.1.2 defines

⟨*hthits*⟩+≡

```
#define MAX_HTDB_LINE   1024
#define MAX_ENTRY_TYPE  30      /* I.e. \page \newcommand \patch ... */
#define MAX_ENTRY_NAME  1024    /* E.g. DifferentialCalculusPage */
#define MAX_COMP_REGEX  1024
```

8.1.3 structs

⟨*hthits*⟩+≡

```
typedef struct pgInfo {
    char name[MAX_ENTRY_NAME];
    long start, size;
} PgInfo ;
```

8.1.4 Local includes

⟨*hthits*⟩+≡

```
#include "debug.h"
#include "hthits.h1"
```

8.1.5 local variables

⟨*hthits*⟩+≡

```
char *progName;
char *pattern;
char *htdbFName;
int gverifydates=0;
regex_t reg_pattern;
```

8.1.6 cmdline

⟨*hthits*⟩+≡

```
void cmdline(int argc,char ** argv) {
    progName = argv[0];
    if (argc != 3) {
        fprintf(stderr, "Usage: %s pattern htdb-file\n", progName);
        exit(1);
    }
    pattern = argv[1];
    htdbFName = argv[2];
}
```

8.1.7 handleHtdb

⟨*hthits*⟩+≡

```
void handleHtdb(void) {
    FILE *htdbFile;
    int c;
    htdbFile = fopen(htdbFName, "r");
    if (htdbFile == NULL)
        badDB();
    while ((c = getc(htdbFile)) != EOF) {
        if (c != '\t')
            badDB();
        ungetc(c, htdbFile);
        handleFile(htdbFile);
    }
    fclose(htdbFile);
}
```

8.1.8 handleFile

⟨*hthits*⟩+≡

```
void handleFile(FILE *htdbFile) {
    static PgInfo *pgInfoV = 0;
    static int pgInfoC = 0;
    char htdbLine[MAX_HTDB_LINE];
    char htfname[MAX_HTDB_LINE];
    time_t httime;
    long htsize;
    struct stat htstat;
    long fstart, fend;
    int rc, i, npages;
    char entname[MAX_ENTRY_NAME], enttype[MAX_ENTRY_TYPE];
    long entoffset, entlineno;
    fgets(htdbLine, MAX_HTDB_LINE, htdbFile);
    sscanf(htdbLine, " %s %ld", htfname, &httime);
    /*
     * 1. Verify file: get size and check modification time.
     */
    rc = stat(htfname, &htstat);
    if (rc == -1) {
        fprintf(stderr, "%s: Cannot access %s\n", progName, htfname);
        exit(1);
    }
    if (gverifydates && (htstat.st_mtime != httime)) {
        fprintf(stderr, "%s: Out of date file %s\n", progName, htfname);
        exit(1);
    }
    htsize = htstat.st_size;
    /*
     * 2. Count the pages in the file.
     */
    npages = 0;
    fstart = ftell(htdbFile);
    fend = ftell(htdbFile);
    while (fgets(htdbLine, MAX_HTDB_LINE, htdbFile) != NULL) {
        if (htdbLine[0] == '\t')
            break;
        if (!strncmp(htdbLine, "\\page", 5))
            npages++;
        fend = ftell(htdbFile);
    }
    /*
     * 3. Find offset and size of each \page (skipping \newcommands etc.)
     */
```

```
    if (npages > pgInfoC) {
        if (pgInfoV)
            free(pgInfoV);

        pgInfoC = npages;
        pgInfoV = (PgInfo *)
            malloc(npages * sizeof(PgInfo));

        if (!pgInfoV) {
            fprintf(stderr, "%s: out of memory\n", progName);
            exit(1);
        }
    }
    fseek(htdbFile, fstart, 0);
    for (i = 0; fgets(htdbLine, MAX_HTDB_LINE, htdbFile) != NULL;) {
        if (htdbLine[0] == '\t')
            break;
        sscanf(htdbLine, "%s %s %ld %ld",
               enttype, entname, &entoffset, &entlineno);
        if (i > 0 && pgInfoV[i - 1].size == -1)
            pgInfoV[i - 1].size = entoffset - pgInfoV[i - 1].start;
        if (!strcmp(enttype, "\\page")) {
            strncpy(pgInfoV[i].name, entname, MAX_ENTRY_NAME);
            pgInfoV[i].start = entoffset;
            pgInfoV[i].size = -1;
            i++;
        }
    }
    if (i > 0 && pgInfoV[i - 1].size == -1)
        pgInfoV[i - 1].size = htsize - pgInfoV[i - 1].start;
    if (i != npages)
        badDB();
    /*
     * 4. Position database input to read next file-description
     */
    fseek(htdbFile, fend, 0);
    /*
     * 5. Process the pages of the file.
     */
    handleFilePages(htfname, npages, pgInfoV);
}
```

8.1.9 handleFilePages

⟨*hthits*⟩+≡

```
void handleFilePages(char *fname, int pgc, PgInfo *pgv) {
    FILE *infile;
    int i;
    infile = fopen(fname, "r");
    if (infile == NULL) {
        fprintf(stderr, "%s: Cannot read file %s\n", progName, fname);
        exit(1);
    }
    for (i = 0; i < pgc; i++)
        handlePage(infile, pgv + i);
    fclose(infile);
}
```

8.1.10 handlePage

⟨*hthits*⟩+≡

```
void handlePage(FILE *infile,PgInfo * pg) {
    static char *pgBuf = 0;
    static int pgBufSize = 0;
    char *title, *body;
    if (pg->size > pgBufSize - 1) {
        if (pgBuf)
            free(pgBuf);
        pgBufSize = pg->size + 20000;
        pgBuf = (char *)malloc(pgBufSize);
        if (!pgBuf) {
            fprintf(stderr,"%s: Out of memory\n", progName);
            exit(1);
        }
    }
    fseek(infile, pg->start, 0);
    fread(pgBuf, pg->size, 1, infile);
    pgBuf[pg->size] = 0;
    splitpage(pgBuf, &title, &body);
    /*untexbuf(title);*/
    untexbuf(body);
#ifdef DEBUG
    printf("-------------- %s -------------\n%s", pg->name, pgBuf);
    printf("============== %s =============\n", title);
    printf("%s", body);
#endif
    searchPage(pg->name, title, body);
}
```

8.1.11 searchPage

⟨*hthits*⟩+≡

```
void searchPage(char *pgname,char * pgtitle,char * pgbody) {
    char *bodyrest;
    regmatch_t match_pos;
    int nhits = 0;
    if (!regexec(&reg_pattern, pgtitle, 1, &match_pos, 0))
        nhits++;
    bodyrest = pgbody;
    while (!regexec(&reg_pattern, bodyrest, 1, &match_pos, 0)) {
        nhits++;
        bodyrest += match_pos.rm_eo;
    }
    if (nhits) {
        printf("\\newsearchresultentry{%d}{%s}",nhits, pgtitle);
        squirt(pgname, strlen(pgname));
        printf("\n");
    }
}
```

8.1.12 squirt

Given string s and length n, output ‘ followed by the first n characters of s with ‘ and newline converted to blanks. This function destructively modifies s.

⟨*hthits*⟩+≡

```
void squirt(char *s, int n) {
    register char *t, *e;
    int c;
    c = s[n];
    for (t = s, e = s + n; t < e; t++)
        if (*t == '‘' || *t == '\n')
            *t = ' ';
    if (s[n] != 0) {
        s[n] = 0;
    }
    printf("{%.*s}", n, s);
    s[n] = c;
}
```

8.1.13 splitpage

Any newlines and separator characters in the title are changed to blanks.

⟨*hthits*⟩+≡

```
void splitpage(char *buf, char **ptitle, char **pbody) {
    int n, depth, tno;
    char *s;
    switch (buf[1]) {
      case 'p':
        tno = 2;
        break;                  /* \page{Name}{Title} */
      case 'b':
        tno = 3;
        break;                  /* \begin{page}{Name}{Title} */
      default:
        fprintf(stderr, "%s: Invalid page format: %s\n", progName, buf);
        exit(1);
    }
    n = 0;
    depth = 0;
    for (s = buf; *s; s++) {
        if (*s == '{')
            if (++depth == 1 && ++n == tno)
                *ptitle = s + 1;
        if (*s == '}')
            if (depth-- == 1 && n == tno) {
                *s = 0;
                *pbody = s + 1;
                break;
            }
    }
}
```

8.1.14 untexbuf

⟨*hthits*⟩+≡

```
void untexbuf(register char *s) {
    register char *d = s;
    while (*s)
        switch (*s) {
          case '\\':
            *d++ = ' ';
            s++;
            if (*s != '%')
                while (isalpha(*s))
                    s++;
            break;
          case '%':
            *d++ = ' ';
            s++;
            while (*s && *s != '\n')
                s++;
            break;
          case '{':
          case '}':
          case '#':
            *d++ = ' ';
            s++;
            break;
          default:
            *d++ = *s++;
        }
    *d = 0;
}
```

8.1.15 badDB

⟨*hthits*⟩+≡

```
void badDB(void) {
    fprintf(stderr, "%s:  bad database file %s\n", progName, htdbFName);
    exit(1);
}
```

8.1.16 regerr

⟨*hthits*⟩+≡

```
void regerr(int code) {
    fprintf(stderr, "%s: regular expression error %d for \"%s\"\n",
            progName, code, pattern);
}
```

8.1.17 main

⟨*hthits*⟩+≡

```
int main(int argc,char ** argv) {
    cmdline(argc, argv);
    regcomp(&reg_pattern, pattern, REG_NEWLINE);
    handleHtdb();
    return(0);
}
```

Chapter 9

The hypertex command

This is the main module of the HyperDoc program. It contains the main routine which initializes all the X stuff, and the tables. Then it passes control over to the main event loop.

9.1 Constants and Headers

9.1.1 System includes

⟨*hypertex*⟩≡

```
#ifdef SGIplatform
#include <bstring.h>
#endif
#include <ctype.h>
#include <fcntl.h>
#include <setjmp.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/errno.h>
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <X11/cursorfont.h>
#include <X11/keysym.h>
#include <X11/X.h>
#include <X11/Xatom.h>
#include <X11/Xresource.h>
```

9.2 structs

⟨*hypertex*⟩+≡

```
typedef struct toke {   /* HyperDoc parser tokens */
  int type;             /* token type. One of those listed below */
  char *id;             /* string value if type == Identifier */
} Token;
```

9.2.1 Local includes

⟨*hypertex*⟩+≡

```
#include "debug.h"
⟨hyper.h⟩

#include "all-hyper-proto.h1"
#include "bsdsignal.h"
#include "bsdsignal.h1"
#include "hterror.h1"
#include "pixmap.h1"
#include "sockio-c.h1"
#include "spadcolors.h"
#include "spadcolors.h1"
#include "util.h1"
```

9.3 structs

⟨*hypertex*⟩+≡

```
typedef struct mr_stack {
    /** The structure for storing parser mode and region **/
    short int fParserMode;
    short int fParserRegion;
    struct mr_stack *fNext;
}   MR_Stack;

typedef struct sock_list {      /* linked list of Sock */
    Sock Socket;
    struct sock_list *next;
}   Sock_List;
```

9.4 defines

⟨*hypertex*⟩+≡

```
#define above(y) ((y) + gWindow->page->scroll_off < gWindow->page->top_scroll_mar
#define AllMode 0

#define BACKCOLOR gControlBackgroundColor
#define below(y) ((y) + gWindow->page->scroll_off >= gWindow->page->bot_scroll_ma
#define BITMAPDEPTH 1
#define bothalf(y) (y/2)
#define bottom_margin 15
#define box_space 3
#define box_width 3
#define BufferSlop      0
#define BUTTGC fControlGC

#define dash_width 5
#define dash_y 4
#define special(c) ((c) == '{' || (c) == '}' || (c) == '#' || (c) == '%' || \
                    (c) == '\\'  || (c) == '[' || (c) == ']' || (c) == '_' || \
                    (c) == ' ' || (c) == '$' || (c) == '~' || (c) == '^' ||  \
                    (c) == '&')

#define punctuation(c) ((c)== '`' || (c) == '\'' || (c) == ','  || \
                        (c) == '.' || (c) == '?' || (c) == '"' || \
                        (c)  == ';' || (c) == ':' || (c) == '-')

#define whitespace(c) ((c) == ' ' || (c) == '\t' || (c) == '\n')
#define delim(c) \
  (whitespace(c) || special(c) || punctuation(c))
#define filedelim(c) \
  (whitespace(c))
#define DependHashSize  20

#define end_page(t) ((t == Page || t == NewCommand ||t == Endpage)?1:0)

#define FORECOLOR gControlForegroundColor
#define funnyEscape(c)  ((c) == '"' ? '\177' : ((c) == '\\' ? '\200' : c))
#define funnyUnescape(c) ((c) == '\177' ? '"' : ((c) == '\200' ? '\\' : c))

#define HTCONDNODE 1 /* unrecognized condition node */
#define htfhSize 100
#define ht_icon_width 40
#define ht_icon_height 40
#define ht_icon_x_hot -1
```

```
#define ht_icon_y_hot -1
static char ht_icon_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0xf7, 0x00, 0x00, 0x00, 0x00, 0xe7, 0x00, 0x00, 0x00,
   0x00, 0xe7, 0x00, 0x00, 0x00, 0x00, 0xe7, 0xef, 0x7b, 0x3c, 0xe7, 0xff,
   0xef, 0x7f, 0x7e, 0xff, 0xff, 0xe7, 0xef, 0xe7, 0xfe, 0xe7, 0x6e, 0xe7,
   0xe7, 0xde, 0xe7, 0x7e, 0xe7, 0xff, 0x0e, 0xe7, 0x3c, 0xe7, 0x07, 0x0e,
   0xe7, 0x3c, 0xf7, 0xcf, 0x0e, 0xf7, 0x18, 0x7f, 0xfe, 0x1f, 0x00, 0x1c,
   0x3f, 0x7c, 0x1f, 0x00, 0x0e, 0x07, 0x00, 0x00, 0x00, 0x0f, 0x07, 0x00,
   0x00, 0x00, 0x87, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x80, 0x3f, 0x00, 0x00, 0x00, 0x80, 0x7f, 0x00, 0x00, 0x00,
   0x00, 0x77, 0x00, 0x00, 0x00, 0x00, 0x77, 0x00, 0x00, 0x00, 0x00, 0x77,
   0x00, 0x00, 0x00, 0x00, 0x77, 0x3e, 0xdc, 0x00, 0x00, 0x77, 0x7f, 0xfe,
   0x00, 0x00, 0xf7, 0xe3, 0xef, 0x00, 0x00, 0xf7, 0xe3, 0xc7, 0x00, 0x00,
   0xf7, 0xe3, 0x07, 0x00, 0x00, 0xf7, 0xe3, 0x07, 0x00, 0x00, 0xf7, 0xe3,
   0xcf, 0x00, 0x80, 0x7f, 0x7f, 0xfe, 0x00, 0x80, 0x3f, 0x3e, 0x7c, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
#define horiz_line_space 3

#define inter_line_space 5
#define inter_word_space 5

#define KEYTYPE 2 /* unrecognized keyword found in lex.c */

#define left_margin 20
#define LinkHashSize 25

#define MaxInputFiles 256
#define MAXLINE      256
#define min(x,y)     ( (x<y)?(x):(y))
#define min_inter_column_space 10
#define mouseBitmap_width 16
#define mouseBitmap_height 16
#define mouseBitmap_x_hot 8
#define mouseBitmap_y_hot 0
static char mouseBitmap_bits[] = {
   0x00, 0x01, 0x00, 0x01, 0x80, 0x02, 0x40, 0x04, 0xc0, 0x06, 0x20, 0x08,
   0x20, 0x08, 0x30, 0x18, 0x50, 0x14, 0x58, 0x34, 0x90, 0x12, 0x20, 0x08,
   0xc0, 0x47, 0x00, 0x21, 0x80, 0x10, 0x00, 0x0f};
#define mouseMask_width 16
#define mouseMask_height 16
static char mouseMask_bits[] = {
   0x00, 0x01, 0x00, 0x01, 0x80, 0x03, 0xc0, 0x07, 0xc0, 0x07, 0xe0, 0x0f,
   0xe0, 0x0f, 0xf0, 0x1f, 0xf0, 0x1f, 0xf8, 0x3f, 0xf0, 0x1f, 0xe0, 0x0f,
```

```
   0xc0, 0x47, 0x00, 0x21, 0x80, 0x10, 0x00, 0x0f};
#define MIN_WINDOW_SIZE 300

#define new_verb_node() \
  resizeVbuf(); \
  *vb = '\0'; \
  curr_node->data.text = allocString(vbuf); \
  curr_node->next = allocNode(); \
  curr_node = curr_node->next; \
  curr_node->type = Newline; \
  curr_node->next = allocNode(); \
  curr_node = curr_node->next; \
  curr_node->type = type; \
  if (*end_string == '\n') es = end_string+1; \
  else es = end_string; \
  size = 0; \
  vb = vbuf;
#define not_in_scroll (!(gDisplayRegion == Scrolling))
#define non_scroll_right_margin_space 20
#define NotSpecial(t) ((t == Quitbutton || t == Returnbutton || \
                       t == Upbutton || t == UnknownPage || \
                       t == UlUnknownPage || t == ErrorPage) ?(0):(1))
#define NoVerticalMode 1
#define numeric(c) ((c >= '0' && c <= '9')?1:0)
#define Numerrors  2

#define paragraph_space 30
#define pix_visible(y, h) \
  (not_in_scroll  || ((y) + gRegionOffset + gWindow->page->scroll_off - h +  \
                     line_height < gWindow->page->bot_scroll_margin  \
                      - gWindow->page->top_scroll_margin  && \
                      (y) + gRegionOffset + gWindow->page->scroll_off >=  0))


#define resizeVbuf()\
  if (size == vbuf_size) { \
     vbuf = resizeBuffer(size + VbufSlop, vbuf, &vbuf_size); \
     vb = vbuf + size; \
  }

#define scroll_right_margin_space 40
#define scroll_top_margin top_margin
#define scrollingTopMargin 5
#define scrollbar_pix_width 3
#define scrollbar_pix_height 3
static char scrollbar_pix_bits[] = {0x00, 0x03, 0x00};
```

```
#define scroller_width 2
#define scroller_height 2
static char scroller_bits[] = {0x01, 0x02};

#define sdown3d_width 21
#define sdown3d_height 21
static char sdown3d_bits[] = {
   0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x15, 0x02, 0x00, 0x0c, 0x51, 0x55, 0x15,
   0xaa, 0xaa, 0x0e, 0x51, 0x5f, 0x15, 0xaa, 0xae, 0x0e, 0x51, 0x5f, 0x15,
   0xaa, 0xae, 0x0e, 0x51, 0x5f, 0x15, 0xea, 0xff, 0x0e, 0xd1, 0x7f, 0x15,
   0xaa, 0xbf, 0x0e, 0x51, 0x5f, 0x15, 0xaa, 0xae, 0x0e, 0x51, 0x55, 0x15,
   0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x15, 0xfe, 0xff, 0x0f, 0x55, 0x55, 0x15,
   0xaa, 0xaa, 0x0a};
#define sdown3dpr_width 21
#define sdown3dpr_height 21
static char sdown3dpr_bits[] = {
   0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x15, 0xfe, 0xff, 0x0f, 0x55, 0x55, 0x11,
   0xae, 0xaa, 0x0a, 0x55, 0x55, 0x11, 0xae, 0xbe, 0x0a, 0x55, 0x5d, 0x11,
   0xae, 0xbe, 0x0a, 0x55, 0x5d, 0x11, 0xae, 0xbe, 0x0a, 0xd5, 0xff, 0x11,
   0xae, 0xff, 0x0a, 0x55, 0x7f, 0x11, 0xae, 0xbe, 0x0a, 0x55, 0x5d, 0x11,
   0xae, 0xaa, 0x0a, 0x55, 0x55, 0x11, 0x06, 0x00, 0x08, 0x55, 0x55, 0x15,
   0xaa, 0xaa, 0x0a};
#define sdown_width  sdown3d_width
#define sdown_height sdown3d_height
#define sdown_bits   sdown3d_bits
#define SimpleMode 2
#define spadcom_indent 30
#define stipple_width 4
#define stipple_height 4
#define storeChar(ch) if (sizeBuf) (*sizeBuf)++; else  *c++ = (ch)
#define storeString(str) for (s=str;*s;s++) {storeChar(*s);}

#define sup3d_width 21
#define sup3d_height 21
static char sup3d_bits[] = {
   0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x15, 0x02, 0x00, 0x0c, 0x51, 0x55, 0x15,
   0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x15, 0xaa, 0xae, 0x0e, 0x51, 0x5f, 0x15,
   0xaa, 0xbf, 0x0e, 0xd1, 0x7f, 0x15, 0xea, 0xff, 0x0e, 0x51, 0x5f, 0x15,
   0xaa, 0xae, 0x0e, 0x51, 0x5f, 0x15, 0xaa, 0xae, 0x0e, 0x51, 0x5f, 0x15,
   0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x15, 0xfa, 0xff, 0x0f, 0x55, 0x55, 0x15,
   0xaa, 0xaa, 0x0a};
#define sup3dpr_width 21
#define sup3dpr_height 21
static char sup3dpr_bits[] = {
   0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x15, 0xfe, 0xff, 0x0f, 0x55, 0x55, 0x11,
   0xae, 0xaa, 0x0a, 0x55, 0x55, 0x11, 0xae, 0xaa, 0x0a, 0x55, 0x5d, 0x11,
```

```
   0xae, 0xbe, 0x0a, 0x55, 0x7f, 0x11, 0xae, 0xff, 0x0a, 0xd5, 0xff, 0x11,
   0xae, 0xbe, 0x0a, 0x55, 0x5d, 0x11, 0xae, 0xbe, 0x0a, 0x55, 0x5d, 0x11,
   0xae, 0xbe, 0x0a, 0x55, 0x55, 0x11, 0x06, 0x00, 0x08, 0x55, 0x55, 0x15,
   0xaa, 0xaa, 0x0a};
#define sup_width  sup3d_width
#define sup_height sup3d_height
#define sup_bits sup3d_bits

#define term_punct_space 5
#define tophalf(y) ((y % 2 == 0)?(y/2):(y/2) + 1)
#define top_margin 5

#define visible(y, h) \
  (not_in_scroll  || ((y) + gRegionOffset + gWindow->page->scroll_off \
                      <= gWindow->scrollheight   && \
                 (y) + gRegionOffset + gWindow->page->scroll_off - (h) >=  0))

#define VbufSlop 10

#define whitespace(c) ((c) == ' ' || (c) == '\t' || (c) == '\n')
```

9.5 externs

⟨*hypertex*⟩+≡

```
extern FILE *cfile;
extern TextNode *curr_node;
extern TextNode *cur_spadcom;   /* spad command being parsed *** */

extern char ebuffer[];
extern jmp_buf env;
extern int example_number;

extern int include_bf;
extern int indent;
extern int item_indent;
extern int item_space;

extern Window gActiveWindow;
extern int gBorderColor;
extern char *gDatabasePath;
extern short int gDisplayRegion;
extern boolean   gEndedPage;
extern short int gExtentRegion;
extern short int gInAxiomCommand;     /* true iff we are in a \spadcommand */
extern boolean gInButton;
extern short int gInDesc;
extern boolean gInIf;
extern short int gInItem; /* true iff we are in a \item */
extern boolean gInItems;
extern int gInInsertMode;
extern short int gInLine;  /* true iff there have been words printed  */
extern boolean gInOptional;
extern short int gInPaste;
extern short int gInSpadsrc;
extern short int gInTable;
extern short int gInVerbatim;
extern HashTable *gLinkHashTable; /* the hash table of active link windows */
extern TextNode *gLineNode;
extern int gNeedIconName;
extern HyperDocPage *gPageBeingParsed;
extern short int gParserMode;
extern short int gParserRegion;
extern int gRegionOffset;
extern int gScrollbarWidth;
extern short int gStringValueOk;
extern GroupItem *gTopOfGroupStack;
extern ItemStack *gTopOfItemStack;
```

```
extern int gTtFontIs850;
extern int gverify_dates;
#ifdef SUN4OS5platform
extern int gethostname(char *, int );
#endif

extern int in_cursor_height;
extern int in_cursor_width;
extern int in_cursor_y;
extern HashTable init_macro_hash;
extern HashTable init_page_hash;
extern HashTable init_patch_hash;
extern int in_next_event;       /* true when in XNextEvent      */
extern int input_file_count;
extern char **input_file_list;

extern jmp_buf jmpbuf;

extern int kill_spad;

extern int line_height; /* space between lines */
extern int line_number;

extern int make_input_file;
extern int make_patch_files;
extern unsigned int ModifiersMask;

extern int need_scroll_down_button;
extern int need_scroll_up_button;
extern int normal_textHeight; /* space between lines */

extern int out_cursor_height;
extern int out_cursor_width;
extern int out_cursor_y;

extern long page_start_fpos;/* tells the character position of the start
                             * of the page, needed to find the current
                             * position when restoring the scanner */
extern ParameterList parameters;
extern int past_line_height;
extern int present_line_height;

extern int received_window_request;     /* true iff Spad wants a pop-up */
extern int right_margin;
extern int right_margin_space;
```

```
extern int scroll_bot;
extern int simple_box_width;
extern int space_width; /* the maximum width of a character */
extern int start_x;
extern int start_y;
extern int still_reading, str_len;

extern int text_x;
extern int text_y;
extern int twheight; /* the height for all windows in the title bar */
extern int twwidth; /* the width for all windows in the title bar */

extern unsigned int UnsupportedModMask;

extern int      vbuff;

extern int word_off_height; /* the diff between text height and */

extern int yOff;               /* y offset for scrolling regions */
```

9.6 local variables

⟨*hypertex*⟩+≡

```
char *active_file_list[MaxInputFiles];

unsigned long bigmask= 0xffffffff;
char buf_for_record_commands[256];

extern FILE *cfile;
static int cur_height = 0;
HyperDocPage *cur_page;
TextNode *curr_node; /* current node being parsed. It is next one filled */
TextNode *cur_spadcom;          /* The current AXIOM command   */

jmp_buf env;
InputBox *end_rb_list;

char *errmess[] =  {
  "place holder",
  "parsing condition node",
  "unrecognized keyword" };

int example_number;
char *ExpectedBeginScroll =
   "Parser Error: Unexpected new page, expecting a begin scroll\n";
char *ExpectedEndScroll =
  "Parser Error: Unexpected new page, expected an end scroll\n";

HyperDocPage *formatpage;

int gActiveColor;
Cursor gActiveCursor;          /* The cursor in active regions           */
XFontStruct *gActiveFont;
Window gActiveWindow;
int gArgc;
char **gArgv;
int gAxiomColor;
XFontStruct *gAxiomFont;
int gBackgroundColor;
int gBfColor;
XFontStruct *gBfFont;
Cursor gBusyCursor;            /* The clock cursor for when I am busy    */
int gBorderColor;     /* The Border Color */
int gControlBackgroundColor;
int gControlForegroundColor;
char *gDatabasePath = NULL;
```

```
short int gDisplayRegion = 0;
int gEmColor;
XFontStruct *gEmFont;
boolean gEndedPage;
short int gExtentRegion;
HashTable gFileHashTable;            /* hash table of HyperDoc files */
HashTable gImageHashTable;           /* hash table for images */
short int gInAxiomCommand;            /* true iff we are in a \spadcommand */
boolean gInButton = FALSE;
short int gInDesc;
boolean gInIf = FALSE;
short int gInItem;               /* true iff we are in a \item */
boolean gInItems = FALSE;
short int gInLine;               /* true iff there have been words printed  */
boolean gInOptional = FALSE;
int gInputBackgroundColor;
XFontStruct *gInputFont;
int gInputForegroundColor;
int gInInsertMode = 0;
short int gInPaste;
short int gInTable;
int gIsAxiomServer = 0; /* true iff HyperDoc is acting as an axiom server  */
int gIsEndOfOutput;            /* set to true when spad has finished output */
int gItColor;
XFontStruct *gItFont;
TextNode *gLineNode;
HashTable *gLinkHashTable;       /* the hash table of active link windows   */
int gmakeRecord_file= 0;       /* true when making record files from ht */
int gNeedIconName = 0;
Cursor gNormalCursor;          /* The normal mouse cursor                */
HDWindow *gParentWindow =NULL; /* the parent window. The one that appears
                                 * when you first start HyperDoc       */
short int gParserMode;           /* Parser mode flag */
short int gParserRegion;         /* Parser Region flag scrolling etc */
int gRegionOffset = 0;
int gRmColor;
XFontStruct *gRmFont;
static HyperLink *gSavedInputAreaLink = NULL;
HashTable gSessionHashTable;   /* hash table of HD windows */
int gSlColor;
short int gStringValueOk;        /* is a string or box value ok */
XFontStruct *gSlFont;
int gSwitch_to_mono=0; /* 1 if at any time we don't have enough colors */
ItemStack *gTopOfItemStack = NULL;
GroupItem *gTopOfGroupStack = NULL;
int gTtFontIs850=0; /* IBM pagecode 850? */
```

```
int gverify_dates = 0;   /* true when we want hypertex to verify ht.db dates */
int gverifyRecord_file = 0;    /* true when verifying record files from ht */
XFontStruct *gTitleFont;
int gTtColor;
XFontStruct *gTtFont;
HDWindow *gWindow = NULL;      /* the current window */
Display *gXDisplay;
int      gXScreenNumber;

HashTable ht_gFileHashTable;

TextNode *if_node = NULL;
char *inactive_file_list[MaxInputFiles];
int include_bf = 0;
int in_cursor_height;
int in_cursor_width;
int in_cursor_y;
int indent;
HashTable init_macro_hash;      /* initial hash table of HD macros */
HashTable init_page_hash;       /* initial hash table of HD pages */
HashTable init_patch_hash;      /* initial hash table of HD patches */
int in_next_event = 0;          /* true when in XNextEvent                 */
int input_file_count;
char **input_file_list;
int item_indent;
int item_space;

int kill_spad = 0;              /* kill spad when finished with paste file */

int line_height;                /* space between lines              */
TextNode *link_node = NULL;

int make_input_file = 0;        /* true when making input files from ht */
int make_patch_files = 0;       /* true when making patch files from ht */
static int maxXvalue = 0;
int MenuServerOpened = 1; /* connected to menu server */
unsigned int ModifiersMask = ShiftMask | LockMask | ControlMask
    | Mod1Mask | Mod2Mask | Mod3Mask
    | Mod4Mask | Mod5Mask;
int motion = 0;

int need_scroll_up_button;
int need_scroll_down_button;
int noop_count;
static char *noopfile = "noop3d.bitmap";
int normal_textHeight;         /* space between lines              */
```

```
int num_active_files = 0;
int num_inactive_files = 0;

int out_cursor_height;
int out_cursor_width;
int out_cursor_y;

ParameterList parameters = NULL;
TextNode *paste_node = NULL;
int past_line_height;
Sock_List *plSock = (Sock_List *) 0;
int present_line_height;
static char *protected_quit;
char *p2sBuf = NULL;
int p2sBufSize = 0;

InputBox *rb_list;
int received_window_request = 0;/* true iff Spad wants a pop-up    */
XrmDatabase rDB;
char *replace_page;            /* true if dynamic page is link to static one */
int ret_val;                    /* The return value from getToken */
int right_margin;
int right_margin_space;

Sock *spadSocket = (Sock *) 0; /* to_server socket for SpadServer */

HyperLink *quitLink;            /** the global link to the quit page ***/

InputItem *save_item;
int scrn;  /* used in spad_colors */
static Pixmap scrollbar_pix = 0;
int gScrollbarWidth = sup_width + 2;
int scroll_bot;
static Pixmap scroller = 0;
static Pixmap sdown = 0;
static Pixmap sdown_pressed = 0;
static GContext server_font;
Sock *sessionServer;           /* socket connecting to session manager */
int simple_box_width;
int space_width;                /* the maximum width of a character */
TextNode *spad_node = NULL;
unsigned long *spadColors;
int start_x;
int start_y;
Pixmap stipple;
static char stipple_bits[] = {0xff, 0xff, 0xff, 0xff};
```

```
static Pixmap sup = 0;
static int supheight = sup_height;
static Pixmap sup_pressed = 0;
static int supwidth = sup_width;

int text_x;
int text_y;
MR_Stack *top_mr_stack = NULL;  /** Declaration for the stack  **/
static XImage *tw1image = NULL;
static XImage *tw2image = NULL;
static XImage *tw3image = NULL;
static XImage *tw4image = NULL;
static XImage *noopimage = NULL;
static char *tw1file  = "exit3d.bitmap";
static char *tw2file  = "help3d.bitmap";
static char *tw3file  = "home3d.bitmap";
static char *tw4file  = "up3d.bitmap";
int twheight;   /* the height for all windows in the title bar */
int twwidth;     /* the width for all windows in the title bar */

unsigned int UnsupportedModMask = LockMask | ControlMask
    | Mod1Mask | Mod2Mask | Mod3Mask
    | Mod4Mask | Mod5Mask;

int word_off_height;            /* the diff between text height and */

int yOff;
```

9.7 The Shared Code

⟨*hypertex*⟩+≡

```
int windowEqual(Window *w1, Window *w2);
int windowCode(Window *w, int size);
CondNode *allocCondnode(void);
char *printToString(TextNode *command);
LineStruct *allocInputline(int size);
void updateInputsymbol(InputItem *sym);
static void drawCursor(InputItem *sym);
static void clearCursorline(InputItem *sym);
void showPage(HyperDocPage *page);
static void clearCursor(InputItem *sym);
static void handleEvent(XEvent * event);
static void createWindow(void);
HyperDocPage *issueServerCommand(HyperLink *link);
HyperDocPage *parsePatch(PasteNode *paste);
static void handleButton(int button, XButtonEvent * event);
HyperDocPage *issueUnixlink(TextNode *node);
static int setWindow(Window window);
static void clearExposures(Window w);
void getNewWindow(void);
HyperDocPage *parsePageFromSocket(void);
static void handleMotionEvent(XMotionEvent *event);
static void initCursorStates(void);
static void makeBusyCursor(HDWindow *window);
static void setErrorHandlers(void);
static void computeBeginItemsExtent(TextNode * node);
static void computeItemExtent(TextNode * node);
static void computeMitemExtent(TextNode *node);
static void endifExtent(TextNode *node);
static void computeIfcondExtent(TextNode *node);
static void computeCenterExtent(TextNode * node);
static void computeBfExtent(TextNode *node);
static void computeEmExtent(TextNode *node);
static void computeItExtent(TextNode *node);
static void computeRmExtent(TextNode *node);
static void computeButtonExtent(TextNode *node);
static void endbuttonExtent(TextNode *node);
static void computePastebuttonExtent(TextNode *node);
static void endpastebuttonExtent(TextNode *node);
static void computePasteExtent(TextNode *node);
static void computeSpadcommandExtent(TextNode *node);
static void computeSpadsrcExtent(TextNode *node);
static void endSpadcommandExtent(TextNode *node);
static void endSpadsrcExtent(TextNode *node);
```

```
static void computeMboxExtent(TextNode *node);
static void computeBoxExtent(TextNode *node);
static void computeIrExtent(TextNode *node);
static void computeImageExtent(TextNode *node);
static void computeTableExtent(TextNode **node);
void computeTitleExtent(HyperDocPage *page);
void computeHeaderExtent(HyperDocPage *page);
void computeFooterExtent(HyperDocPage * page);
void computeScrollingExtent(HyperDocPage *page);
void startNewline(int distance, TextNode * node);
static void centerNodes(TextNode * begin_node, TextNode * end_node);
static void makeBusyCursors(void);
void initExtents(void);
void initTitleExtents(HyperDocPage * page);
static int textHeight1(TextNode * node, int Ender);
static int Xvalue(TextNode * node);
void insertBitmapFile(TextNode * node);
void insertPixmapFile(TextNode * node);
void computeFormPage(HyperDocPage *page);
static int windowHeight(HyperDocPage *page);
static void formHeaderExtent(HyperDocPage *page);
static void formFooterExtent(HyperDocPage *page);
static void formScrollingExtent(HyperDocPage *page);
void pushGroupStack(void);
void emTopGroup(void);
void rmTopGroup(void);
void bfTopGroup(void);
void pushActiveGroup(void);
void pushSpadGroup(void);
void initTopGroup(void);
void centerTopGroup(void);
HDWindow *allocHdWindow(void);
static void makeTheInputFile(UnloadedPage *page);
static void makeInputFileFromPage(HyperDocPage *page);
static int inListAndNewer(char *inputFile, char *htFile);
static void makeInputFileList(void);
static void sendCommand(char *command,int com_type);
static void printPaste(FILE *pfile,char *realcom,char *command,
            char *pagename,int com_type);
static void printGraphPaste(FILE *pfile,char *realcom,
                  char *command,char *pagename,int com_type);
HyperDocPage *allocPage(char *name);
static void setNameAndIcon(void);
static int getBorderProperties(void);
static void openWindow(Window w);
static void setSizeHints(Window w);
```

```
static void getGCs(HDWindow *window);
static void ingItColorsAndFonts(void);
void changeText(int color, XFontStruct *font);
static int getColor(char *name, char *class, int def, Colormap *map);
static void mergeDatabases(void);
void toggleInputBox(HyperLink *link);
static void clearRbs(InputBox *list);
void changeInputFocus(HyperLink *link);
void pushItemStack(void);
void clearItemStack(void);
void popItemStack(void);
void handleKey(XEvent *event);
FILE *findFp(FilePosition fp);
TextNode *allocNode(void);
static void getParameterStrings(int number,char * macro_name);
void toggleRadioBox(HyperLink *link);
void freeHdWindow(HDWindow *w);
static void dontFree(void  *link);
static void freeCond(CondNode *cond);
void freePage(HyperDocPage *page);
static void freeDepend(SpadcomDepend *sd);
static void freeInputBox(InputBox *box);
static void freePastebutton(TextNode *node, short int des);
static void freePastearea(TextNode *node, short int des);
void freeInputItem(InputItem *sym, short int des);
void freeInputList(InputItem *il);
static void freeRadioBoxes(RadioBoxes *radio);
void freeButtonList(ButtonList *bl);
void loadPage(HyperDocPage *page);
static HyperDocPage *formatPage(UnloadedPage *ulpage);
void parseFromString(char *str);
static void parsePage(HyperDocPage *page);
void parseHyperDoc(void);
char *windowId(Window w);
static void startScrolling(void);
static void startFooter(void);
static void endAPage(void);
static void parseReplacepage(void);
void readHtDb(HashTable *page_hash, HashTable *macro_hash,
                HashTable *patch_hash);
static void readHtFile(HashTable *page_hash, HashTable *macro_hash,
                        HashTable *patch_hash, FILE *db_fp, char *dbFile);
void makeSpecialPages(HashTable *pageHashTable);
void addDependencies(void);
void parserError(char *str);
void parseInputstring(void);
```

```
void parseSimplebox(void);
ImageStruct *insertImageStruct(char *filename);
static void addBoxToRbList(char *name,InputBox *box);
static int checkOthers(InputBox *list);
static void insertItem(InputItem *item);
void parsePaste(void);
void parsePastebutton(void);
static void loadPatch(PatchStore *patch);
void parseIfcond(void);
static void parseCondnode(void);
static void parseHasreturnto(void);
void parseNewcond(void);
void parseSetcond(void);
void parseBeginItems(void);
void parseItem(void);
void parseMitem(void);
void parseVerbatim(int type);
void parseInputPix(void);
void parseCenterline(void);
void parseCommand(void);
void parseButton(void);
void parseSpadcommand(TextNode *spad_node);
void parseSpadsrc(TextNode *spad_node);
void parseEnv(TextNode *node);
void parseValue1(void);
void parseValue2(void);
void parseTable(void);
void parseBox(void);
void parseMbox(void);
void parseFree(void);
void parseHelp(void);
static int readHot(FILE *fd,char Line[],int *x_hot,int *y_hot);
static int readWandH(FILE *fd,unsigned int *width,unsigned int *height);
static int ch(int height);
static void changeWindowBackgroundPixmap(Window window, Pixmap pixmap);
void showText(TextNode *node, int Ender);
static void showLink(TextNode *node);
static void showPaste(TextNode *node);
static void showPastebutton(TextNode *node);
static void showInput(TextNode *node);
static void showSimpleBox(TextNode *node);
static void showSpadcommand(TextNode *node);
static void showImage(TextNode *node, GC gc);
void issueSpadcommand(HyperDocPage *page, TextNode *command,
                      int immediate, int type);
static void sendPile(Sock *sock,char * str);
```

```
static void issueDependentCommands(HyperDocPage *page,
                                   TextNode *command,int type);
static void markAsExecuted(HyperDocPage *page, TextNode *command,int type);
static void startUserBuffer(HyperDocPage *page);
static void clearExecutionMarks(HashTable *depend_hash);
Sock *acceptMenuConnection(Sock *server_sock);
static void acceptMenuServerConnection(HyperDocPage *page);
char *printToString1(TextNode *command,int * sizeBuf);
void issueUnixcommand(TextNode *node);
void serviceSessionSocket(void);
static void switchFrames(void);
void sendLispCommand(char *command);
void escapeString(char *s);
void unescapeString(char *s);
static void closeClient(int pid);
char *printSourceToString(TextNode *command);
char *printSourceToString1(TextNode *command,int * sizeBuf);
static void readTitleBarImages(void);
void displayPage(HyperDocPage *page);
void parseRadiobox(void);
void parseRadioboxes(void);
void dumpToken(char *caller, Token t);
void printNextTenTokens(void);
int getToken(void);
int keywordType(void);
int popGroupStack(void);
int initTopWindow(char *name);
int initFormWindow(char *name, int cols);
int totalWidth(TextNode * node, int Ender);
int textWidth(TextNode * node, int Ender);
int maxX(TextNode * node, int Ender);
int textHeight(TextNode * node, int Ender);
int isIt850(XFontStruct *fontarg);
int getFilename(void);
int issueServerpaste(TextNode *command);
int issueUnixpaste(TextNode *node);

char *vbuf = NULL;
int vbuf_size = 0;
```

⟨*hypertex shared code*⟩
⟨*hashCopyEntry*⟩
⟨*hashCopyTable*⟩
⟨*dbFileOpen*⟩
⟨*htperror*⟩
⟨*dumpToken*⟩

9.8 Code

9.8.1 sigusr2Handler

SIGUSR2 is raised by the spadbuf program when it is done with the current command

⟨*hypertex*⟩+≡
```
void sigusr2Handler(int sig) {
  gIsEndOfOutput = 1;
  return ;
}
```

9.8.2 sigcldHandler

Why were we waiting after the child had already died? Because we don't want zombies

⟨*hypertex*⟩+≡
```
void sigcldHandler(int sig) {
  int x;
  wait(&x);
}
```

9.8.3 cleanSocket

Clean up spad sockets on exit.

⟨*hypertex*⟩+≡
```
void cleanSocket(void) {
    char name[256];
    make_server_name(name, MenuServerName);
    unlink(name);
}
```

9.8.4 initHash

Initializes the hash table for Files, and Windows

⟨*hypertex*⟩+≡

```
static void initHash(void) {
    hashInit(&gFileHashTable,
             FileHashSize,
             (EqualFunction)stringEqual,
             (HashcodeFunction) stringHash);
    hashInit(&gSessionHashTable,
             SessionHashSize,
             (EqualFunction) windowEqual,
             (HashcodeFunction) windowCode);
    hashInit(&gImageHashTable,
             ImageHashSize,
             (EqualFunction) stringEqual,
             (HashcodeFunction) stringHash);
}
```

9.8.5 initPageStructs

Initialize the HyperDoc page hierarchy data structures

⟨*hypertex*⟩+≡

```
void initPageStructs(HDWindow *w) {
    int i;
    w->fMemoStackIndex = 0;
    for (i = 0; i < MaxMemoDepth; i++) {
        w->fMemoStack[i] = NULL;
        w->fDownLinkStackTop[i] = 0;
    }
    w->fDownLinkStackIndex = 0;
    for (i = 0; i < MaxDownlinkDepth; i++)
        w->fDownLinkStack[i] = NULL;
}
```

9.8.6 checkArguments

⟨*hypertex*⟩+≡

```
static void checkArguments(void) {
  int i;
   /*
   * Now check the command line arguments, to see if I am supposed to be a
   * server or not
   */
  for (i = 1; i < gArgc; i++) {
    if (gArgv[i][0] == '-')
      switch (gArgv[i][1]) {
      case 'p':
        gverify_dates=1;
        break;
      case 's':
        if (!MenuServerOpened) {
          fprintf(stderr, "(HyperDoc) Server already in use.\n");
          exit(-1);
        }
        gIsAxiomServer = 1;
        break;
      case 'i':
        if (gArgv[i][2] == 'p')
          make_patch_files = 1;
        make_input_file = 1;
        input_file_list = gArgv + i + 1;
        input_file_count = gArgc - i - 1;
        break;
      case 'k':
        kill_spad = 1;
        break;
      case 'r':
        if (gArgv[i][2] == 'm')
          gmakeRecord_file=1;
        else if (gArgv[i][2] == 'v')
          gverifyRecord_file=1;
        else
         fprintf(stderr, "(HyperDoc) v or m must follow -r\n");
        input_file_list = gArgv + i + 1;
        input_file_count = gArgc - i - 1;
        break;
      default:
        fprintf(stderr, "(HyperDoc) Unexpected Command Line Argument ");
        fprintf(stderr,"%s\n", gArgv[i]);
        fprintf(stderr, "           Usage: hypertex [-s]\n");
```

```
          break;
        }
    }
  }
```

9.8.7 makeServerConnections

⟨*hypertex*⟩+≡

```
static void makeServerConnections(void) {
    int i, wait_time;
    /*
     * Try to open the menuserver socket, if I can not, then set a flag
     */
    if (open_server(MenuServerName) == -2) {
       fprintf(stderr, "(HyperDoc) Warning: Not connected to AXIOM Server!\n");
       MenuServerOpened = 0;
    }
    else {
     /* In order to allow hyperdoc restarts from the console we clean up
      * the socket on exit */
       atexit(&cleanSocket);
       MenuServerOpened = 1;
    }
    /*
     * If I have opened the MenuServer socket, then I should also try to open
     * the SpadServer socket, so I can send stuff right to SPAD.
     */
    if (MenuServerOpened) {
        /*
         * If I am a ht server, then I should not continue on unless I
         * establish some sort of connection
         */

        /*
         * Modified on 11/20 so that it prints an error message every ten for
         * ten tries at opeing the socket. If it fails all ten times, it
         * gives up and exits.
         */
        if (!gIsAxiomServer)
            wait_time = 2;
        else
            wait_time = 1000;
        for (i = 0, spadSocket = NULL; i < 2 && spadSocket == NULL; i++) {
            spadSocket = connect_to_local_server(SpadServer,
                                                 MenuServer, wait_time);
            if (gIsAxiomServer && spadSocket == NULL)
                fprintf(stderr,
                    "(HyperDoc) Error opening AXIOM server. Retrying ...\n");
            else
                i = 11;
        }
```

```
        if (! spadSocket) {
            fprintf(stderr, "(HyperDoc) Couldn't connect to AXIOM server!\n");
            if (!gIsAxiomServer)
                MenuServerOpened = 0;
            else {
                fprintf(stderr,"(HyperDoc) Cannot connect to AXIOM server\n");
                exit(-1);
            }
        }
        else {
            /*
             * Do the same thing for the SessionServer
             */
            for (i = 0, sessionServer = NULL; i < 2 && sessionServer == NULL
                 ; i++) {
                sessionServer =
                    connect_to_local_server(SessionServer, MenuServer,
                                            wait_time);
                if (gIsAxiomServer && sessionServer == NULL) {
                    fprintf(stderr,
                     "(HyperDoc) Error opening SessionServer, Retrying ...\n");
                }
                else
                    i = 11;
            }
            if (sessionServer == NULL) {
                fprintf(stderr, "(HyperDoc) Connection attempt to session ");
                fprintf(stderr,"manager timed out.\n");
                if (gIsAxiomServer) {
                    fprintf(stderr,
                    "(HyperDoc) Server unable to connect to session server\n");
                    exit(-1);
                }
                else {
                    MenuServerOpened = 0;
                }
            }
        }
    }
}
```

9.9 Condition Handling

9.9.1 insertCond

This routine creates a new cond node and inserts it into the current cond table

⟨*hypertex*⟩+≡

```
void insertCond(char *label, char *cond) {
    CondNode *condnode = (CondNode *) hashFind(gWindow->fCondHashTable, label);
    if (condnode) {
        fprintf(stderr, "Error: \\%s is declared twice \n", label);
        printPageAndFilename();
        jump();
    }
    condnode = allocCondnode();
    condnode->label = halloc(strlen(label) + 1, "Condnode->label");
    condnode->cond = halloc(strlen(cond) + 1, "Condnode->cond");
    strcpy(condnode->label, label);
    strcpy(condnode->cond, cond);
    hashInsert(gWindow->fCondHashTable, (char *) condnode, condnode->label);
}
```

9.9.2 changeCond

⟨*hypertex*⟩+≡

```
void changeCond(char *label, char *newcond) {
    CondNode *condnode = (CondNode *) hashFind(gWindow->fCondHashTable, label);
    if (condnode == NULL) {
        fprintf(stderr, "Error: Tried to set an uncreated cond %s\n", label);
    }
    else {
        free(condnode->cond);
        condnode->cond = halloc(strlen(newcond) + 1, "Condnode->cond");
        strcpy(condnode->cond, newcond);
    }
}
```

9.9.3 checkMemostack

⟨*hypertex*⟩+≡

```
static int checkMemostack(TextNode *node) {
    char *buffer;
    int stackp = gWindow->fMemoStackIndex;
    int found = 0;
    HyperDocPage *page;
    buffer = printToString(node->data.node);
    /*
     * Once we have done that much, search down the stack for the
     * proper page
     */
    while (!found && stackp > 0) {
        page = gWindow->fMemoStack[--stackp];
        if (!strcmp(page->name, buffer))
            found = 1;
    }
    return found;
}
```

9.9.4 checkCondition

Checks the condition presented and returns a 1 or a 0.

⟨*hypertex*⟩+≡

```
int checkCondition(TextNode *node) {
    CondNode *cond;
    InputBox *box;
    int ret_val;
    switch (node->type) {
      case Cond:
        cond = (CondNode *) hashFind(gWindow->fCondHashTable, node->data.text);
        if (!strcmp("0", cond->cond))
            return 0;
        else
            return 1;
      case Boxcond:
        box = (InputBox *) hashFind(gWindow->page->box_hash, node->data.text);
        return (box->picked);
      case Haslisp:
        if (spadSocket != NULL) {
            ret_val = send_int(spadSocket, TestLine);
            return (ret_val + 1);
        }
        else
            return 0;
      case Hasup:
        return need_up_button;
      case Hasreturn:
        return gWindow->fMemoStackIndex;
      case Hasreturnto:
        return (checkMemostack(node));
      case Lastwindow:
        return(gSessionHashTable.num_entries == 1 || gParentWindow == gWindow);
      default:
        return 0;
    }
}
```

9.10 Dialog Handling

9.10.1 redrawWin

⟨*hypertex*⟩+≡
```
static void redrawWin(void) {
    XUnmapSubwindows(gXDisplay, gWindow->fMainWindow);
    XUnmapSubwindows(gXDisplay, gWindow->fScrollWindow);
    XFlush(gXDisplay);
    showPage(gWindow->page);
}
```

9.10.2 mystrncpy

Copies the characters from buff1 to buff2 starting at position buff2+n and buff1+n

⟨*hypertex*⟩+≡
```
static char *mystrncpy(char *buff1, char *buff2, int n) {
    int i;
    for (i = n - 1; i >= 0; i--)
        *(buff1 + i) = *(buff2 + i);
    return buff2;
}
```

9.10.3 incLineNumbers

⟨*hypertex*⟩+≡
```
static void incLineNumbers(LineStruct *line) {
    for (; line != NULL; line = line->next)
        line->line_number++;
}
```

9.10.4 decLineNumbers

⟨*hypertex*⟩+≡

```
static void decLineNumbers(LineStruct *line) {
    for (; line != NULL; line = line->next)
        line->line_number--;
    return;
}
```

9.10.5 decreaseLineNumbers

⟨*hypertex*⟩+≡

```
static void decreaseLineNumbers(LineStruct *line, int am) {
    for (; line != NULL; line = line->next)
        line->line_number -= am;
}
```

9.10.6 overwriteBuffer

⟨*hypertex*⟩+≡

```
static void overwriteBuffer(char *buffer, InputItem *item) {
    LineStruct *newline;
    LineStruct *addline = item->curr_line;
    /*int bufflen = strlen(buffer);*/
    int nl = 0;
    int cursor_y;
    int size = item->size;
    /* add a single character */
    cursor_y = (addline->line_number - 1) * line_height;
    if (addline->buff_pntr == size) {
        clearCursor(item);
        if (addline->len <= size) {
            nl = 1;
            addline->buffer[size] = '_';
            addline->buffer[size + 1] = 0;
            addline->len = size + 1;
            newline = (LineStruct *) allocInputline(size + 2);
            newline->line_number = addline->line_number + 1;
            incLineNumbers(addline->next);
            newline->next = addline->next;
            newline->prev = addline;
            if (addline->next)
                addline->next->prev = newline;
            addline->next = newline;
            item->num_lines++;
            cursor_y += line_height;
            item->curr_line = addline = newline;
        }
        else {
            item->curr_line = addline = addline->next;
        }
        addline->len = 1;
        addline->buff_pntr = 1;
        addline->buffer[0] = buffer[0];
    }
    else {
        addline->buffer[addline->buff_pntr] = buffer[0];
        clearCursor(item);
        if (++addline->buff_pntr > addline->len)
            addline->len++;
    }
    /* now set up the current line */
    if (item->curr_line->buff_pntr >= item->size &&
```

```
        item->curr_line->next != NULL && !item->curr_line->next->len) {
        /* I should actually be on the next line */
        item->curr_line->buffer[item->size] = '_';
        item->curr_line->len = item->size + 1;
        XDrawString(gXDisplay, item->win, gWindow->fInputGC, start_x,
                    cursor_y + start_y,
                    addline->buffer,
                    addline->len);
        item->curr_line = item->curr_line->next;
        item->curr_line->buff_pntr = 0;
        item->curr_line->changed = 1;
    }
    if (!nl) {
        XDrawString(gXDisplay, item->win, gWindow->fInputGC, start_x,
                    cursor_y + start_y,
                    addline->buffer,
                    addline->len);
        drawCursor(item);
    }
    else
        redrawWin();
}

/*
 */
```

9.10.7 moveSymForward

This routine takes the current line and moves it num forward. The only way I have to move any other lines forward is if this line has length ¿ size

⟨*hypertex*⟩+≡

```
static int moveSymForward(LineStruct *line, int num, int size,
                          InputItem *sym) {
    LineStruct *newline;
    int diff;
    int nl = 0;
    if (line->len > size) {
        nl = moveSymForward(line->next, num, size, sym);
        strncpy(line->next->buffer,
                &line->buffer[sym->size - num], line->len);
        strncpy(&line->buffer[num],
                line->buffer, num);
        line->changed = 1;
        return nl;
    }
    else {
        if (line->len + num > size) {
            diff = line->len + num - size;
            newline = allocInputline(size);
            newline->len = diff;
            newline->line_number = line->line_number++;
            incLineNumbers(line->next);
            sym->num_lines++;
            newline->next = line->next;
            newline->prev = line;
            if (line->next)
                line->next->prev = newline;
            line->next = newline;
            strncpy(newline->buffer, &line->buffer[size - diff], diff);
            strncpy(&line->buffer[num], line->buffer, num);
            line->buffer[size] = '_';
            line->buffer[size + 1] = 0;
            line->len = size + 1;
            return 1;
        }
        else {
            strncpy(&line->buffer[num], line->buffer, line->len);
            line->len += num;
            line->changed = 1;
            return 0;
        }
```

```
    }
}
```

9.10.8 clearCursorline

⟨*hypertex*⟩+≡

```
static void clearCursorline(InputItem *sym) {
    XCharStruct extents;
    int dir, asc, des;
    int cursor_y;
    XTextExtents(gInputFont, sym->curr_line->buffer,
                 sym->curr_line->buff_pntr,
                 &dir, &asc, &des, &extents);
    cursor_y = (sym->curr_line->line_number - 1) * line_height;
    sym->cursor_x = start_x + extents.width;
    XClearArea(gXDisplay, sym->win, sym->cursor_x, cursor_y,
               gWindow->width, line_height, False);
    XDrawString(gXDisplay, sym->win, gWindow->fInputGC, start_x,
                cursor_y + start_y, sym->curr_line->buffer,
                sym->curr_line->len);
}
```

9.10.9 insertBuffer

⟨*hypertex*⟩+≡

```
static void insertBuffer(char *buffer, InputItem *sym) {
    /*int num = strlen(buffer);*/
    LineStruct *line = sym->curr_line;
    LineStruct *newline;
    int nl = 0;
    int size = sym->size;
    if (line->len < size) {
        /* they will all fit where I am so just copy them forward */
        line->len++;
        mystrncpy(&(line->buffer[line->buff_pntr + 1]),
                  &(line->buffer[line->buff_pntr]),
                  line->len - line->buff_pntr + 1);
        line->buffer[line->buff_pntr] = buffer[0];
        clearCursorline(sym);
        line->buff_pntr++;
        drawCursor(sym);
        return;
    }
    if (line->len > sym->size) {
        nl = moveSymForward(line->next, 1, size, sym);
        if (line->buff_pntr > size) {
            line->changed = 1;
            line = line->next;
            line->buffer[0] = buffer[0];
            line->len++;
            line->buff_pntr = 1;
            line->changed = 1;
        }
        else {
            line->next->buffer[0] = line->buffer[size - 1];
            line->changed = 1;
            strncpy(&line->buffer[line->buff_pntr + 1],
                &line->buffer[line->buff_pntr], size - line->buff_pntr - 1);
            line->buffer[line->buff_pntr++] = buffer[0];
            line->changed = 1;
            if (line->buff_pntr >= size) {
                sym->curr_line = line->next;
                sym->curr_line->buff_pntr = 0;
            }
        }
    }
    else {
        nl = 1;
```

```
        newline = allocInputline(size);
        newline->line_number = line->line_number + 1;
        incLineNumbers(line->next);
        sym->num_lines++;
        newline->next = line->next;
        newline->prev = line;
        if (line->next)
            line->next->prev = newline;
        line->next = newline;
        /*
         * was line->buff_pntr++;
         */
        if (line->buff_pntr >= size) {
            /* we are the leaders of the line */
            newline->buff_pntr = 1;
            newline->buffer[0] = buffer[0];
            newline->len = 1;
            sym->curr_line = newline;
        }
        else {
            /* we are not the leaders */
            newline->buffer[0] = line->buffer[size - 1];
            newline->len = 1;
            strncpy(&line->buffer[line->buff_pntr + 1],
                    &line->buffer[line->buff_pntr], size - line->buff_pntr);
            if (line->buff_pntr < size - 1) {
                line->buffer[line->buff_pntr++] = buffer[0];
            }
            else {
                line->buffer[line->buff_pntr] = buffer[0];
                newline->buff_pntr = 0;
                sym->curr_line = newline;
            }
        }
        line->buffer[size] = '_';
        line->buffer[size + 1] = 0;
        line->len = size + 1;
    }
    if (nl)
        redrawWin();
    else
        updateInputsymbol(sym);
}
```

9.10.10 addBufferToSym

⟨*hypertex*⟩+≡

```
void addBufferToSym(char *buffer,InputItem *sym) {
    if (gInInsertMode)
        insertBuffer(buffer, sym);
    else
        overwriteBuffer(buffer, sym);
}
```

9.10.11 drawInputsymbol

⟨*hypertex*⟩+≡

```
void drawInputsymbol(InputItem *sym) {
    int y_spot = start_y;
    LineStruct *cline;
    XCharStruct extents;
    int dir, asc, des;
#if 0
    int cursor_y;
    cursor_y = (sym->curr_line->line_number - 1) * line_height;
#endif
    XClearWindow(gXDisplay, sym->win);

    XTextExtents(gInputFont, sym->curr_line->buffer,
                 sym->curr_line->buff_pntr,
                 &dir, &asc, &des, &extents);
    sym->cursor_x = start_x + extents.width;
    /*
     * While the list of input strings is not NULL, I should just keep
     * drawing them
     */
    for (cline = sym->lines; cline != NULL;
         cline = cline->next, y_spot += line_height) {
        /* Now I should draw the initial string ** */
        cline->changed = 0;
        XDrawString(gXDisplay, sym->win, gWindow->fInputGC, start_x, y_spot,
                    cline->buffer,
                    cline->len);
    }
    if (gWindow->page->currentItem == sym)
        drawCursor(sym);
}
```

9.10.12 updateInputsymbol

⟨*hypertex*⟩+≡

```
void updateInputsymbol(InputItem *sym) {
    int y_spot = start_y;
    LineStruct *cline;
    XCharStruct extents;
    int dir, asc, des;
    /*int cleared = 0;*/
    int clear_y;
    int clear_width;
    int clear_height;
#if 0
    int cursor_y;
    cursor_y = (sym->curr_line->line_number - 1) * line_height;
#endif
    clear_width = (sym->size + 1) * gInputFont->max_bounds.width + 10;
    clear_height = line_height;
    clear_y = 0;
    XTextExtents(gInputFont, sym->curr_line->buffer,
                 sym->curr_line->buff_pntr,
                 &dir, &asc, &des, &extents);
    sym->cursor_x = start_x + extents.width;
    /*
     * While the list of input strings is not NULL, I should just keep
     * drawing them
     */
    for (cline = sym->lines; cline != NULL;
         cline = cline->next, y_spot += line_height, clear_y += line_height)
        /* Now I should draw the initial string ** */
        if (cline->changed) {
            cline->changed = 0;
            XClearArea(gXDisplay, sym->win, 0, clear_y,
                       clear_width, clear_height, False);
            XDrawString(gXDisplay, sym->win, gWindow->fInputGC, start_x,
                        y_spot, cline->buffer, cline->len);
        }
    drawCursor(sym);
}
```

9.10.13 drawCursor

⟨*hypertex*⟩+≡

```
static void drawCursor(InputItem *sym) {
    int cursor_y;
    XCharStruct extents;
    int dir, asc, des;
    cursor_y = (sym->curr_line->line_number - 1) * line_height;
    XTextExtents(gInputFont, sym->curr_line->buffer,
                 sym->curr_line->buff_pntr,
                 &dir, &asc, &des, &extents);
    sym->cursor_x = start_x + extents.width;
    /* now draw the cursor */
    if (gInInsertMode) {
        XFillRectangle(gXDisplay, sym->win, gWindow->fInputGC,
                       sym->cursor_x,
                       out_cursor_y + cursor_y,
                       out_cursor_width,
                       out_cursor_height);
        /* Now draw the character currently under the cursor */
        XDrawString(gXDisplay, sym->win, gWindow->fCursorGC,
                    sym->cursor_x, cursor_y + start_y,
                    &sym->curr_line->buffer[sym->curr_line->buff_pntr],
                    1);
    }
    else
        XFillRectangle(gXDisplay, sym->win, gWindow->fInputGC,
                       sym->cursor_x,
                       in_cursor_y + cursor_y,
                       in_cursor_width,
                       in_cursor_height);
}
```

9.10.14 moveCursorHome

⟨*hypertex*⟩+≡

```
static void moveCursorHome(InputItem *sym) {
    LineStruct *trace = sym->curr_line;
    /* now move the cursor  to the beginning of the current line */
    clearCursor(sym);
    for (; trace && trace->prev && trace->prev->len > sym->size;)
        trace = trace->prev;
    sym->curr_line = trace;
    trace->buff_pntr = 0;
    drawCursor(sym);
}
```

9.10.15 moveCursorEnd

⟨*hypertex*⟩+≡

```
static void moveCursorEnd(InputItem *sym) {
    LineStruct *trace = sym->curr_line;
    /* now move the cursor  to the beginning of the current line */
    clearCursor(sym);
    for (; trace && trace->next && trace->len > sym->size;)
        trace = trace->next;
    sym->curr_line = trace;
    trace->buff_pntr = trace->len;
    drawCursor(sym);
}
```

9.10.16 void moveCursorForward

⟨*hypertex*⟩+≡

```
static void moveCursorForward(InputItem *sym) {
    if (sym->curr_line->buff_pntr == sym->curr_line->len &&
        !sym->curr_line->next) {
        BeepAtTheUser();
        return;
    }
    if (sym->curr_line->buff_pntr == sym->curr_line->len ||
        sym->curr_line->buff_pntr == sym->size - 1)
    {
        /* I have to move down to a new line */
        if (sym->curr_line->next == NULL) {
            /* now where to move */
            BeepAtTheUser();
            return;
        }
        /* move down line */
        clearCursor(sym);
        sym->curr_line = sym->curr_line->next;
        sym->curr_line->buff_pntr = 0;
    }
    else {
        clearCursor(sym);
        sym->curr_line->buff_pntr++;
    }
    drawCursor(sym);
}
```

9.10.17 moveCursorDown

⟨*hypertex*⟩+≡

```
static void moveCursorDown(InputItem *sym) {
    int bp = sym->curr_line->buff_pntr;
    /*int size = sym->size;*/
    LineStruct *trace;
    /* get to the end of the current line */
    for (trace = sym->curr_line; trace->len > sym->size; trace = trace->next)
        ;
    if (!trace->next)
        BeepAtTheUser();
    else {
        clearCursor(sym);
        sym->curr_line = trace->next;
        if (bp > sym->curr_line->len)
            sym->curr_line->buff_pntr = sym->curr_line->len;
        else
            sym->curr_line->buff_pntr = bp;
        drawCursor(sym);
    }
}
```

9.10.18 moveCursorUp

⟨*hypertex*⟩+≡

```
static void moveCursorUp(InputItem *sym) {
    int bp = sym->curr_line->buff_pntr;
    /*int size = sym->size;*/
    LineStruct *trace;
    /* get to the end of the current line */
    for (trace = sym->curr_line;
         trace->prev && trace->prev->len > sym->size;
         trace = trace->prev)
            ;
    if (!trace->prev)
        BeepAtTheUser();
    else {
        clearCursor(sym);
        sym->curr_line = trace->prev;
        if (bp > sym->curr_line->len)
            sym->curr_line->buff_pntr = sym->curr_line->len;
        else
            sym->curr_line->buff_pntr = bp;
        drawCursor(sym);
    }
}
```

9.10.19 clearCursor

⟨*hypertex*⟩+≡

```
static void clearCursor(InputItem *sym) {
    XCharStruct extents;
    int dir, asc, des;
    int cursor_y;
    XTextExtents(gInputFont, sym->curr_line->buffer,
                 sym->curr_line->buff_pntr,
                 &dir, &asc, &des, &extents);
    cursor_y = (sym->curr_line->line_number - 1) * line_height;
    sym->cursor_x = start_x + extents.width;
    XClearArea(gXDisplay, sym->win, sym->cursor_x, cursor_y,
               in_cursor_width, line_height, False);
    XDrawString(gXDisplay, sym->win, gWindow->fInputGC,
                start_x, cursor_y + start_y,
                sym->curr_line->buffer,
                sym->curr_line->len);
}
```

9.10.20 moveCursorBackward

⟨*hypertex*⟩+≡

```
static void moveCursorBackward(InputItem *sym) {
    if (sym->curr_line->buff_pntr == 0) {
        if (sym->curr_line->prev == NULL) {
            /* now where to move */
            BeepAtTheUser();
            return;
        }
        else {
            clearCursor(sym);
            /* move up to the previous line */
            sym->curr_line = sym->curr_line->prev;
            if (sym->curr_line->len > sym->size)
                sym->curr_line->buff_pntr = sym->size - 1;
            else
                sym->curr_line->buff_pntr = sym->curr_line->len;
        }
    }
    else {            /* just slide back a char. on the current line */
        clearCursor(sym);
        sym->curr_line->buff_pntr--;
    }
    drawCursor(sym);
}
```

9.10.21 moveRestBack

⟨*hypertex*⟩+≡

```
static char moveRestBack(LineStruct *line, int size) {
    char c = '\000';
    if (line != NULL && line->len != 0)
        c = line->buffer[0];
    else
        return c;
    while (line->next != NULL && line->len > size) {
        strncpy(line->buffer, &(line->buffer[1]), size - 1);
        line->buffer[size - 1] = line->next->buffer[0];
        line->changed = 1;
        line = line->next;
    }
    /*
     * once I get here I should be one the last line, so I can just copy all
     * the characters back one and then return from whence I came
     */
    if (line->len > 0) {
        line->changed = 1;
        if (line->len > 1)
            strncpy(line->buffer, &(line->buffer[1]), line->len - 1);
        line->buffer[--line->len] = 0;
        if (line->len == 0) {
            /* I have to fix the previous line */
            line->prev->len = size;
            line->prev->buffer[size] = 0;
        }
    }
    return c;
}
```

9.10.22 deleteRestOfLine

⟨*hypertex*⟩+≡

```
static void deleteRestOfLine(InputItem *sym) {
    LineStruct *curr_line = sym->curr_line;
    LineStruct *line=NULL;
    LineStruct *trash;
    LineStruct *trace;
    int num_changed = 0, i;
    if (curr_line->len > sym->size) {
        for (line = curr_line->next, num_changed = 0;
             line != NULL && line->len > 0 && line->len > sym->size;
             line = line->next, num_changed++) {
            line->len = 0;
            line->buffer[0] = 0;
            line->changed = 1;
        }
        num_changed++;
    }
    if (num_changed == 0 && curr_line->buff_pntr == curr_line->len) {
        if (curr_line->len == 0 && curr_line->next) {
            curr_line->next->prev = curr_line->prev;
            if (curr_line->prev)
                curr_line->prev->next = curr_line->next;
            else
                sym->lines = curr_line->next;
            decLineNumbers(curr_line->next);
            sym->num_lines--;
            sym->curr_line = curr_line->next;
            sym->curr_line->buff_pntr = 0;
            free(curr_line->buffer);
            free(curr_line);
            redrawWin();
        }
        else
            BeepAtTheUser();
        return;
    }
    curr_line->len = curr_line->buff_pntr;
    /* curr_line->buffer[curr_line->len] = NULL; */
    for (i = curr_line->len; i <= sym->size + 2; i++)
        curr_line->buffer[i] = 0;
    curr_line->changed = 1;
    if (num_changed) {
        /* I should get rid of all these lines */
        trace = curr_line->next;
```

```
        curr_line->next = line->next;
        if (line->next)
            line->next->prev = curr_line;
        for (; trace && trace != line->next;) {
            trash = trace;
            trace = trace->next;
            free(trash->buffer);
            free(trash);
        }
        decreaseLineNumbers(curr_line->next, num_changed);
        sym->num_lines -= num_changed;
        redrawWin();
    }
    else
        updateInputsymbol(sym);
}
```

9.10.23 backOverEoln

⟨*hypertex*⟩+≡

```
static void backOverEoln(InputItem *sym) {
    /*
     * This routine is very similar to a tough enter except it starts
     * combining lines with sym->curr_line->pre
     */
    char buff[1024];
    LineStruct *trace;
    LineStruct *last = NULL;
    char *tr = buff;
    int bp;
    int size = sym->size;
    /* copy all the stuff into the buffer */
    for (trace = sym->curr_line;
         trace->len > sym->size; trace = trace->next)
        for (bp = 0; bp < size; bp++)
            *tr++ = trace->buffer[bp];
    /* copy the last line */
    for (bp = 0; bp < trace->len; bp++)
        *tr++ = trace->buffer[bp];
    trace->len = 0;
    *tr = 0;
    /* Now that I have the buffer, let's put it back where it belongs. */
    last = trace;
    for (trace = sym->curr_line; trace != last; trace = trace->next);
    trace = sym->curr_line = sym->curr_line->prev;
    trace->buff_pntr = trace->len;
    trace->changed = 1;
    for (bp = trace->len, tr = buff; bp < size && *tr; bp++)
        trace->buffer[bp] = *tr++;
    if (!*tr) {
        trace->len = bp;
    }
    else {
        trace->len = size + 1;
        trace->buffer[size] = '_';
        trace->buffer[size + 1] = 0;
        for (trace = trace->next; *tr;) {
            for (bp = 0; bp < size && *tr; bp++)
                trace->buffer[bp] = *tr++;
            if (*tr) {
                trace->len = size + 1;
                trace->changed = 1;
                trace->buffer[size + 1] = 0;
```

```
                trace->buffer[size] = '_';
                trace = trace->next;
            }
            else {
                trace->len = bp;
                trace->buffer[bp] = 0;
            }
        }
    }
    /* Now once I am here, let me see if I can bag a line */
    if (last->len == 0) {
        /* rid myself of this line */
        last->prev->next = last->next;
        if (last->next)
            last->next->prev = last->prev;
        decLineNumbers(last->next);
        sym->num_lines--;
        free(last->buffer);
        free(last);
        redrawWin();
    }
    else
        updateInputsymbol(sym);
}
```

9.10.24 moveBackOneChar

⟨*hypertex*⟩+≡

```
static int moveBackOneChar(InputItem *sym) {
    char c = '\000', d = '\000';
    int dl = 0;
    /* This routine moves all the characters back one */
    LineStruct *line = sym->curr_line;
    if (line->len > sym->size)
        c = moveRestBack(line->next, sym->size);
    line->changed = 1;
    if (line->buff_pntr == 0) { /* I am at the front of the line */
        if (line->prev == 0) {
            BeepAtTheUser();
            return 0;
        }
        else if (line->prev->len <= sym->size) {
            backOverEoln(sym);
            return 1;
        }
        else if (line->len > 0) {
            d = line->buffer[0];
            if (line->len <= sym->size) {
                strncpy(line->buffer, &(line->buffer[1]), line->len - 1);
                if (c == 0) {
                    line->len--;
                    line->buffer[line->len] = 0;
                }
                else
                    line->buffer[line->len - 1] = c;
            }
            else {
                strncpy(line->buffer, &(line->buffer[1]), sym->size - 2);
                if (c == 0) {
                    line->buffer[sym->size - 1] = 0;
                    line->len--;
                }
                else {
                    line->buffer[sym->size - 1] = c;
                }
            }
        }
        else {
            /* the line is just going to be thrown away */
            if (line->next)
                line->next->prev = line->prev;
```

```
            line->prev->next = line->next;
            decLineNumbers(line->next);
            sym->num_lines--;
            free(line->buffer);
            free(line);
            dl = 1;
        }
        c = d;
        sym->curr_line = line = line->prev;
        line->changed = 1;
        line->buff_pntr = sym->size;
    }
    if (line->len <= sym->size) {
        strncpy(&line->buffer[line->buff_pntr - 1],
                &(line->buffer[line->buff_pntr]),
                line->len - line->buff_pntr);
        if (c == 0)
            line->buffer[--line->len] = 0;
        else
            line->buffer[line->len - 1] = c;
    }
    else {
        strncpy(&(line->buffer[line->buff_pntr - 1]),
                &(line->buffer[line->buff_pntr]),
                sym->size - line->buff_pntr);
        if (c == 0) {
            line->buffer[sym->size - 1] = 0;
            line->len = sym->size - 1;
        }
        else {
            if (line->next->len == 0) {
                line->buffer[sym->size] = 0;
                line->len = sym->size;
            }
            line->buffer[sym->size - 1] = c;
        }
    }
    line->buff_pntr--;
    if (dl)
        redrawWin();
    else
        updateInputsymbol(sym);
    return 1;
}
```

9.10.25 backOverChar

⟨*hypertex*⟩+≡

```
static void backOverChar(InputItem *sym) {
    if (moveBackOneChar(sym))
        updateInputsymbol(sym);
}
```

9.10.26 deleteEoln

⟨*hypertex*⟩+≡

```
static void deleteEoln(InputItem *sym) {
    /* much the same as back_over eoln except my perspective has changed */
    char buff[1024];
    LineStruct *trace;
    LineStruct *last = 0;
    char *tr = buff;
    int bp;
    int size = sym->size;
    /* copy all the stuff into the buffer */
    for (trace = sym->curr_line->next;
         trace->len > sym->size; trace = trace->next)
        for (bp = 0; bp < size; bp++)
            *tr++ = trace->buffer[bp];
    /* copy the last line */
    for (bp = 0; bp < trace->len; bp++)
        *tr++ = trace->buffer[bp];
    trace->len = 0;
    *tr = 0;
    /* Now that I have the buffer, let's put it back where it belongs. */
    last = trace;
    trace = sym->curr_line;
    trace->changed = 1;
    for (bp = trace->len, tr = buff; bp < size && *tr; bp++)
        trace->buffer[bp] = *tr++;
    if (!*tr)
        trace->len = bp;
    else {
        trace->len = size + 1;
        trace->buffer[size] = '_';
        trace->buffer[size + 1] = 0;
        for (trace = trace->next; *tr;) {
            for (bp = 0; bp < size && *tr; bp++)
                trace->buffer[bp] = *tr++;
            if (*tr) {
                trace->len = size + 1;
                trace->changed = 1;
                trace->buffer[size + 1] = 0;
                trace->buffer[size] = '_';
                trace = trace->next;
            }
            else {
                trace->len = bp;
                trace->buffer[bp] = 0;
```

```
            }
        }
    }
    /* Now once I am here, let me see if I can bag a line */
    if (last->len == 0) {
        /* rid myself of this line */
        last->prev->next = last->next;
        if (last->next)
            last->next->prev = last->prev;
        decLineNumbers(last->next);
        sym->num_lines--;
        free(last->buffer);
        free(last);
        redrawWin();
    }
    else
        updateInputsymbol(sym);
}
```

9.10.27 deleteOneChar

⟨*hypertex*⟩+≡

```
static int deleteOneChar(InputItem *sym) {
    char c = '\000';
    /* This routine moves all the characters back one */
    LineStruct *line = sym->curr_line;
    if (line->len > sym->size)
        c = moveRestBack(line->next, sym->size);
    if (c == 0 && line->len == line->buff_pntr) {
        if (line->next == 0) {
            BeepAtTheUser();
            return 0;
        }
        else {
            deleteEoln(sym);
            return 1;
        }
    }
    /*
     * let me just try to do the copy and put the stupid character c if it
     * exists at the end
     */
    if (line->len <= sym->size) {
        strncpy(&line->buffer[line->buff_pntr],
                &(line->buffer[line->buff_pntr + 1]),
                line->len - line->buff_pntr);
        if (c == 0)
            line->buffer[--line->len] = 0;
        else
            line->buffer[line->len - 1] = c;
    }
    else {
        strncpy(&(line->buffer[line->buff_pntr]),
                &(line->buffer[line->buff_pntr + 1]),
                sym->size - line->buff_pntr);
        if (c == 0) {
            line->buffer[sym->size - 1] = 0;
            line->len = sym->size - 1;
        }
        else {
            if (line->next->len == 0) {
                line->buffer[sym->size] = 0;
                line->len = sym->size;
            }
            line->buffer[sym->size - 1] = c;
```

```
        }
    }
    line->changed = 1;
    return 1;
}
```

9.10.28 deleteChar

⟨*hypertex*⟩+≡

```
static void deleteChar(InputItem *sym) {
    if (deleteOneChar(sym))
        updateInputsymbol(sym);
}
```

9.10.29 toughEnter

This routine takes all the characters from the current cursor on, and copies them into a temp buffer, from which they are recopied back starting at the next line.

⟨*hypertex*⟩+≡

```
static void toughEnter(InputItem *sym) {
    char buff[1024];
    LineStruct *trace;
    LineStruct *last = 0;
    LineStruct *newline;
    char *tr = buff;
    int bp = sym->curr_line->buff_pntr;
    int size = sym->size;
    /* Copy the stuff from the current line */
    for (; bp < size; bp++)
        *tr++ = sym->curr_line->buffer[bp];
    /* now get the stuff from the rest of the lines */
    for (trace = sym->curr_line->next;
         trace->len > sym->size; trace = trace->next)
        for (bp = 0; bp < size; bp++)
            *tr++ = trace->buffer[bp];
    /* copy the last line */
    for (bp = 0; bp < trace->len; bp++)
        *tr++ = trace->buffer[bp];
    *tr = 0;
    /* Now that I have the buffer, let's put it back where it belongs. */
    last = trace;
    trace = sym->curr_line;
    trace->len = trace->buff_pntr;
    trace->buffer[trace->len] = 0;
    trace->changed = 1;
    tr = buff;
    for (trace = trace->next; trace != last; trace = trace->next) {
        for (bp = 0; bp < size; bp++)
            trace->buffer[bp] = *tr++;
        trace->len = size + 1;
        trace->buffer[size + 1] = 0;
        trace->buffer[size] = '_';
        trace->changed = 1;
    }
    /* Once I am here, I should be able to copy this last line */
    for (bp = 0; bp < size && *tr; bp++)
        trace->buffer[bp] = *tr++;
    trace->changed = 1;
```

```
    /* If I still have more to copy, then do so onto a new line */
    if (*tr) {
        trace->len = size + 1;
        trace->buffer[size + 1] = 0;
        trace->buffer[size] = '_';
        newline = allocInputline(size);
        sym->num_lines++;
        newline->line_number = last->line_number + 1;
        incLineNumbers(newline->next);
        for (bp = 0; *tr; bp++)
            newline->buffer[bp] = *tr++;
        newline->len = bp;
        newline->next = last->next;
        newline->prev = last;
        last->next = newline;
        if (newline->next)
            newline->next->prev = newline;
    }
    else {
        trace->len = bp;
        trace->buffer[bp] = 0;
    }
    /* Last but not least change the curr_line */
    sym->curr_line = sym->curr_line->next;
    sym->curr_line->buff_pntr = 0;
}
```

9.10.30 enterNewLine

At this point the user has hit a return. Let me just be naive, and take everything from the current spot on, and put it on a new line

⟨*hypertex*⟩+≡

```
static void enterNewLine(InputItem *sym) {
    LineStruct *newline;
    LineStruct *trace;
    LineStruct *prev;
    LineStruct *line = sym->curr_line;
    int bp = line->buff_pntr;
    int l = line->len;
    int size = sym->size;
    if (bp == 0) {
        if (line->prev->len > size) {
            /* just add a return to the end of the last line */
            prev = line->prev;
            prev->buffer[size] = 0;
            prev->len = size;
            prev->changed = 1;
        }
        else {
            newline = allocInputline(size);
            newline->next = sym->curr_line;
            newline->prev = sym->curr_line->prev;
            line->prev = newline;
            sym->num_lines++;
            if (newline->prev)
                newline->prev->next = newline;
            newline->len = newline->buff_pntr = 0;
            newline->line_number = line->line_number;
            if (sym->curr_line == sym->lines)
                sym->lines = newline;
            for (trace = newline->next; trace != 0; trace = trace->next)
                trace->line_number++;
        }
    }
    else if (bp == size &&
             line->len > size) {
        /* line->next; */
        newline = allocInputline(size);
        if (line->next)
            line->next->prev = newline;
        newline->prev = sym->curr_line;
        line->next = newline;
```

```
        newline->len = 0;
        newline->buff_pntr = 0;
        sym->num_lines++;
        sym->curr_line = newline;
        newline->line_number = newline->prev->line_number + 1;
        for (trace = newline->next; trace != 0; trace = trace->next)
            trace->line_number++;
    }
    else {
        if (line->len > size)
            toughEnter(sym);
        else {
            newline = allocInputline(size);
            strncpy(newline->buffer, &sym->curr_line->buffer[bp], l - bp);
            sym->curr_line->len = bp;
            sym->curr_line->buffer[bp] = '\0';
            newline->next = sym->curr_line->next;
            if (sym->curr_line->next)
                sym->curr_line->next->prev = newline;
            newline->prev = sym->curr_line;
            sym->curr_line->next = newline;
            newline->len = l - bp;
            newline->buff_pntr = 0;
            sym->num_lines++;
            sym->curr_line = newline;
            newline->line_number = newline->prev->line_number + 1;
            for (trace = newline->next; trace != 0; trace = trace->next)
                trace->line_number++;
        }
    }
    redrawWin();
}
```

9.10.31 dialog

⟨*hypertex*⟩+≡

```
void dialog(XEvent *event, KeySym keysym, char *buffer) {
    InputItem *item;
    item = gWindow->page->currentItem;
    if (item == 0) {
        if (!((keysym >= XK_Shift_L) && (keysym <= XK_Hyper_R)))
            /** if something other than a modifier key was hit **/
            BeepAtTheUser();
        return;
    }
    /* First check if the user had hit an enter key */
    if ((keysym == XK_Return) || (keysym == XK_KP_Enter))
        enterNewLine(item);
    /* Else did the user actual type a character I can understand */
    else if (((keysym >= XK_KP_Space) && (keysym <= XK_KP_9))
             || ((keysym >= XK_space) && (keysym <= XK_asciitilde)))
    {
        /* only handle normal keys */
        if (event->xkey.state & UnsupportedModMask)
            BeepAtTheUser();
        else
            addBufferToSym(buffer, item);
    }
    else if ((keysym >= XK_Shift_L) && (keysym <= XK_Hyper_R))
        ;
    /*
     * do nothing, a modifier was hit
     */
    else if ((keysym >= XK_F2) && (keysym <= XK_F35)) {
        /*
         * A function key was hit
         */
        if (strlen(buffer) == 0)
            BeepAtTheUser();
        else
            /* If I got characters then add it to the buffer */
            addBufferToSym(buffer, item);
    }
    else
        switch (keysym) {
          case XK_Escape:
            if (event->xkey.state & ModifiersMask)
                BeepAtTheUser();
            else {
```

```
            moveCursorHome(item);
            deleteRestOfLine(item);
        }
        break;
    case XK_F1:
        if (event->xkey.state & ModifiersMask)
            BeepAtTheUser();
        else {
            gWindow->page->helppage = allocString(InputAreaHelpPage);
            helpForHyperDoc();
        }
        break;
    case XK_Up:
        if (event->xkey.state & ModifiersMask)
            BeepAtTheUser();
        else
            moveCursorUp(item);
        break;
    case XK_Down:
        if (event->xkey.state & ModifiersMask)
            BeepAtTheUser();
        else
            moveCursorDown(item);
        break;
    case XK_Delete:
        if (event->xkey.state & ModifiersMask)
            BeepAtTheUser();
        else
            deleteChar(item);
        break;
    case XK_BackSpace:
        if (event->xkey.state & ModifiersMask)
            BeepAtTheUser();
        else
            backOverChar(item);
        break;
    case XK_Left:
        if (event->xkey.state & ModifiersMask)
            BeepAtTheUser();
        else
            moveCursorBackward(item);
        break;
    case XK_Right:
        if (event->xkey.state & ModifiersMask)
            BeepAtTheUser();
        else
```

```
                moveCursorForward(item);
            break;
        case XK_Insert:
            if (event->xkey.state & ModifiersMask)
                BeepAtTheUser();
            else {
                gInInsertMode = ((gInInsertMode) ? (0) : (1));
                item->curr_line->changed = 1;
                updateInputsymbol(item);
            }
            break;
        case XK_Home:
            if (event->xkey.state & ModifiersMask)
                BeepAtTheUser();
            else
                moveCursorHome(item);
            break;
        case XK_End:
            if (event->xkey.state & ControlMask)
                /* delete from here to the end of the line */

                deleteRestOfLine(item);
            else if (event->xkey.state & ModifiersMask)
                BeepAtTheUser();
            else
                moveCursorEnd(item);
            break;
        default:
            BeepAtTheUser();
            break;
    }
}
```

9.11 Format and Display a page

Display is performed in two steps. First the page is formatted assuming that we have an infinitely long window. In this stage we compute and store the coordinates of every text node. Next the page is actually drawn on the screen. In this process we use the value of page-¿y_off as an offset into the scrolling region to compute what is actually to be displayed on the page.

9.11.1 showPage

⟨*hypertex*⟩+≡

```
void showPage(HyperDocPage *page) {
    XWindowChanges wc;
    int doShowScrollBars = 1;
    initTopGroup();
    /* Clear the areas so we can rewrite the page */
    XClearWindow(gXDisplay, gWindow->fMainWindow);
    XClearWindow(gXDisplay, gWindow->fScrollWindow);
    /* Free the active button list */
    freeButtonList(page->s_button_list);
    page->s_button_list = NULL;
    freeButtonList(page->button_list);
    page->button_list = NULL;
    /* The compute the text extents  */
    computeTitleExtent(page);
    computeHeaderExtent(page);
    computeFooterExtent(page);
    computeScrollingExtent(page);
    /*
     * Now that we have all the extents computed, reconfigure and map the
     * scroll window
     */
    if (page->scrolling) {
        int width, height;
        calculateScrollBarMeasures();
        wc.x = 0;
        wc.y = page->top_scroll_margin + scroll_top_margin;
        wc.height = gWindow->scrollheight;
        if (gWindow->page->scrolling->height <= gWindow->scrollheight) {
            gWindow->page->scroll_off = 0;
            wc.width = gWindow->width;
        }
        else
            wc.width = gWindow->width - gScrollbarWidth;
        getScrollBarMinimumSize(&width, &height);
        if (height > wc.height) {
            wc.height = gWindow->scrollheight = 1;
            doShowScrollBars = 0;
        }
        else
            gWindow->scrollwidth = wc.width;
        if (doShowScrollBars) {
            XConfigureWindow(gXDisplay, gWindow->fScrollWindow,
                             CWX | CWY | CWHeight | CWWidth, &wc);
```

```
            XMapWindow(gXDisplay, gWindow->fScrollWindow);
        }
        else {
            XUnmapWindow(gXDisplay, gWindow->fScrollWindow);
            hideScrollBars(gWindow);
        }
    }
    /* clear the group stack */
    while (popGroupStack() >= 0)
        ;
    /* Now start displaying all the text */
    gWindow->fDisplayedWindow = gWindow->fMainWindow;
    gRegionOffset = 0;
    yOff = 0;
    gDisplayRegion = Header;
    showText(page->header->next, Endheader);
    if (doShowScrollBars && page->scrolling) {
        /* Show the footer  */
        if (page->footer->next) {
            gDisplayRegion = Footer;
            gRegionOffset = gWindow->page->bot_scroll_margin +
                (!((gWindow->page->pageFlags & NOLINES)) ?
                                        ((int) line_height / 2) : (0));
            showText(page->footer->next, Endfooter);
            /* Show the scrolling region */
            if (page->scrolling->next)
                gDisplayRegion = Scrolling;
            gRegionOffset = 0;
            gWindow->fDisplayedWindow = gWindow->fScrollWindow;
            yOff = gWindow->page->scroll_off;
            showText(page->scrolling->next, Endscrolling);
            showScrollBars(gWindow);
        }
        drawScrollLines();
    }
    if (gTopOfItemStack != NULL) {
        fprintf(stderr, "warning: unbalanced \\beginitems .. \\enditems\n");
        gTopOfItemStack = NULL;
    }
    showTitleBar();
    XFlush(gXDisplay);
}
```

9.11.2 exposePage

⟨*hypertex*⟩+≡

```
void exposePage(HyperDocPage *page) {
    int width, height, doShowScrollBars = 1;
    initTopGroup();
    /*
     * Now start displaying all the text
     */
    yOff = 0;
    gWindow->fDisplayedWindow = gWindow->fMainWindow;
    gRegionOffset = 0;
    gDisplayRegion = Header;
    showText(page->header->next, Endheader);
    getScrollBarMinimumSize(&width, &height);
    /*
     * Now see If I have anything left to display
     */
    if (page->scrolling) {
        if (page->footer->next) {
            gDisplayRegion = Footer;
            gRegionOffset = gWindow->page->bot_scroll_margin +
                (!((gWindow->page->pageFlags & NOLINES)) ?
                                        ((int) line_height / 2) : (0));
            showText(page->footer->next, Endfooter);
        }
        if (height > gWindow->scrollheight) {
            gWindow->scrollheight = 1;
            doShowScrollBars = 0;
            XUnmapWindow(gXDisplay, gWindow->fScrollWindow);
            hideScrollBars(gWindow);
        }
        if (page->scrolling->next) {
            gRegionOffset = page->top_scroll_margin;
            gDisplayRegion = Scrolling;
            gRegionOffset = 0;
            gWindow->fDisplayedWindow = gWindow->fScrollWindow;
            yOff = gWindow->page->scroll_off;
            showText(page->scrolling->next, Endscrolling);
            if (doShowScrollBars)
                showScrollBars(gWindow);
        }
        if (doShowScrollBars)
            drawScrollLines();
    }
    showTitleBar();
```

```
    XFlush(gXDisplay);
}
```

9.11.3 scrollPage

⟨*hypertex*⟩+≡

```
void scrollPage(HyperDocPage *page) {
    initTopGroup();
    /* free the active button list */
    freeButtonList(page->s_button_list);
    page->s_button_list = NULL;
    /** Clear the scrolling area */
    XUnmapSubwindows(gXDisplay, gWindow->fScrollWindow);
    gDisplayRegion = Scrolling;
    gRegionOffset = 0;
    gWindow->fDisplayedWindow = gWindow->fScrollWindow;
    yOff = gWindow->page->scroll_off;
    showText(page->scrolling->next, Endscrolling);
    moveScroller(gWindow);
    XFlush(gXDisplay);
}
```

9.11.4 pastePage

⟨*hypertex*⟩+≡

```
void pastePage(TextNode *node) {
    int width, height;
    int old_off = gWindow->page->scroll_off;
    /* free the active button list */
    freeButtonList(gWindow->page->s_button_list);
    gWindow->page->s_button_list = NULL;
    freeButtonList(gWindow->page->button_list);
    gWindow->page->button_list = NULL;
    XUnmapSubwindows(gXDisplay, gWindow->fScrollWindow);
    initTopGroup();
    /* recompute the extent of the scrolling region */
    computeScrollingExtent(gWindow->page);
    calculateScrollBarMeasures();
    getScrollBarMinimumSize(&width, &height);
    /* get ready to show the scrolling area */
    gRegionOffset = 0;
    yOff = gWindow->page->scroll_off;
    gDisplayRegion = Scrolling;
    gWindow->fDisplayedWindow = gWindow->fScrollWindow;
    if (gWindow->page->scroll_off == old_off) {
        XClearArea(gXDisplay, gWindow->fScrollWindow, 0,
                   node->y - line_height + gRegionOffset + yOff,
                   gWindow->width,
                   gWindow->scrollheight - node->y + line_height - yOff,
                   False);
        XFlush(gXDisplay);
    }
    else
        XClearWindow(gXDisplay, gWindow->fScrollWindow);
    showText(gWindow->page->scrolling->next, Endscrolling);
    XFlush(gXDisplay);
    hideScrollBars(gWindow);
    if (height > gWindow->scrollheight) {
        gWindow->scrollheight = 1;
        XUnmapWindow(gXDisplay, gWindow->fScrollWindow);
    }
    else {
        showScrollBars(gWindow);
        drawScrollLines();
        /* moveScroller(); */
    }
    XFlush(gXDisplay);
}
```

9.12 Event Handling

This is the main X loop. It keeps grabbing events. Since the only way the window can die is through an event, it never actually end. One of the subroutines it calls is responsible for killing everything.

9.12.1 mainEventLoop

⟨*hypertex*⟩+≡

```
void mainEventLoop(void) {
    XEvent event;
    int  Xcon;
    fd_set rd, dum1, dum2;
    motion = 0;
    gActiveWindow = -1;
    setErrorHandlers();
    Xcon = ConnectionNumber(gXDisplay);
    while (1) {
        /*fprintf(stderr,"event:mainEventLoop: loop top\n");*/
        while (gSessionHashTable.num_entries == 0)
            pause();
        /* XFlush(gXDisplay);      */
        if (!motion)
            initCursorStates();
        motion = 0;
        if (!spadSocket == 0) {
            FD_ZERO(&rd);
            FD_ZERO(&dum1);
            FD_ZERO(&dum2);
            FD_CLR(0, &dum1);
            FD_CLR(0, &dum2);
            FD_CLR(0, &rd);
            FD_SET(spadSocket->socket, &rd);
            FD_SET(Xcon, &rd);
            if (!sessionServer == 0) {
                FD_SET(sessionServer->socket, &rd);
            }
            if (XEventsQueued(gXDisplay, QueuedAlready)) {
                XNextEvent(gXDisplay, &event);
                handleEvent(&event);
            }
            else {
              select(FD_SETSIZE,(void *)&rd,(void *)&dum1,(void *)&dum2,NULL);
              if (FD_ISSET(Xcon, &rd) ||
                  XEventsQueued(gXDisplay, QueuedAfterFlush)) {
                    XNextEvent(gXDisplay, &event);
                    handleEvent(&event);
                }
                else if FD_ISSET
                    (spadSocket->socket, &rd)
                    /*
                     * Axiom Socket do what handleEvent does The 100 is
```

```
                     * $SpadStuff in hypertex.boot
                     */
                {
                    if (100 == get_int(spadSocket)) {
                        setWindow(gParentWindow->fMainWindow);
                        makeBusyCursors();
                        getNewWindow();
                    }
                }
                /*
                 * Session Socket Telling us about the death of a spadbuf
                 * (plus maybe more later) serviceSessionSocket in
                 * spadint.c
                 */
                else
                 if (sessionServer && FD_ISSET(sessionServer->socket, &rd)) {
                    serviceSessionSocket();
                 }
            }
        }
        else {
            XNextEvent(gXDisplay, &event);
            handleEvent(&event);
        }
    }
}
```

9.12.2 handleEvent

⟨*hypertex*⟩+≡

```
static void handleEvent(XEvent * event) {
    XWindowAttributes wa;
/*    fprintf(stderr,"event:handleEvent entered\n");*/
    setWindow(event->xany.window);
    if (event->type == MotionNotify) {
/*        fprintf(stderr,"event:handleEvent type=MotionNotify\n");*/
        handleMotionEvent((XMotionEvent *)event);
        motion = 1;
        return;
    }
    makeBusyCursors();
    switch (event->type) {
      case DestroyNotify:
/*        fprintf(stderr,"event:handleEvent type=DestroyNotify\n");*/
        break;
      case Expose:
/*        fprintf(stderr,"event:handleEvent type=Expose\n");*/
        XGetWindowAttributes(gXDisplay, gWindow->fMainWindow, &wa);
        if ((gWindow->width == 0 && gWindow->height == 0) ||
            (wa.width != gWindow->width || wa.height != gWindow->height)) {
            gWindow->width = wa.width;
            gWindow->height = wa.height;
            displayPage(gWindow->page);
            gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;
        }
        else                    /** just redraw the thing **/
            exposePage(gWindow->page);
        XFlush(gXDisplay);
        clearExposures(gWindow->fMainWindow);
        clearExposures(gWindow->fScrollWindow);
        break;
      case ButtonPress:
/*        fprintf(stderr,"event:handleEvent type=ButtonPress\n");*/
        handleButton(event->xbutton.button, (XButtonEvent *)event);
        XFlush(gXDisplay);
        if (gWindow) {
            while (XCheckTypedWindowEvent(gXDisplay, gWindow->fMainWindow,
                                          Expose, event));
            while (XCheckTypedWindowEvent(gXDisplay, gWindow->fScrollWindow,
                                          Expose, event));
        }
        break;
      case KeyPress:
```

```
/*        fprintf(stderr,"event:handleEvent type=KeyPress\n");*/
        handleKey(event);
        if (gWindow) {
            while (XCheckTypedWindowEvent(gXDisplay, gWindow->fMainWindow,
                                          Expose, event));
            while (XCheckTypedWindowEvent(gXDisplay, gWindow->fScrollWindow,
                                          Expose, event));
        }
        break;
      case MapNotify:
/*        fprintf(stderr,"event:handleEvent type=MapNotify\n");*/
        createWindow();
        break;

      case SelectionNotify:
/*        fprintf(stderr,"event:handleEvent type=SelectionNotify\n");*/
        /* this is in response to a previous request in an input area */
        if ( gSavedInputAreaLink ) {
            XSelectionEvent *pSelEvent;
            Atom dataProperty;
            pSelEvent = (XSelectionEvent *) event;
            dataProperty = XInternAtom(gXDisplay, "PASTE_SELECTION", False);
            /* change the input focus */

        /*  changeInputFocus(gSavedInputAreaLink); */

            /* try to get the selection as a window property */

            if ( pSelEvent->requestor == gWindow->fMainWindow &&
                 pSelEvent->selection == XA_PRIMARY &&
            /*   pSelEvent->time      == CurrentTime && */
                 pSelEvent->target    == XA_STRING &&
                 pSelEvent->property == dataProperty )
            {
                Atom actual_type;
                int  actual_format;
                unsigned long nitems, leftover;
                char *pSelection = NULL;

                if (Success == XGetWindowProperty(gXDisplay,
                    gWindow->fMainWindow,
                    pSelEvent->property, 0L, 100000000L, True,
                    AnyPropertyType, &actual_type, &actual_format,
                    &nitems, &leftover, (unsigned char **) &pSelection) )
                {
                    char *pBuffer;
```

```
                    InputItem *item = gSavedInputAreaLink->reference.string;

                    for (pBuffer = pSelection; *pBuffer; ++pBuffer)
                        addBufferToSym(pBuffer, item);

                    XFree(pSelection);
                }
            }

            /* clear the link info */

            gSavedInputAreaLink = NULL;
        }
        break;

      default:
/*        fprintf(stderr,"event:handleEvent type=default\n");*/
        break;
    }

}
```

9.12.3 createWindow

⟨*hypertex*⟩+≡

```
static void createWindow(void) {
    XWindowAttributes wa;
    XGetWindowAttributes(gXDisplay, gWindow->fMainWindow, &wa);
    gWindow->width = wa.width;
    gWindow->height = wa.height;
    displayPage(gWindow->page);
    gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;
    /* then select for the events I normally would like to catch */
    XSelectInput(gXDisplay, gWindow->fMainWindow, ButtonPress | KeyPressMask |
                 PointerMotionMask |
                 ExposureMask /* | EnterWindowMask | LeaveWindowMask */ );
    XSelectInput(gXDisplay, gWindow->fScrollWindow, ExposureMask);
}

/*
 */
```

9.12.4 quitHyperDoc

This routine is called when the quitbutton is hit. For the moment I am just going to leave it all behind.

⟨*hypertex*⟩+≡

```
void quitHyperDoc(void) {
    HyperDocPage *page;
    if (gSessionHashTable.num_entries == 1 || gParentWindow == gWindow) {
        if (!strcmp(gWindow->page->name, "ProtectedQuitPage")){
          exitHyperDoc();
        }
        page =
         (HyperDocPage *)hashFind(gWindow->fPageHashTable,"ProtectedQuitPage");
        if (page == NULL) {
            fprintf(stderr, "Unknown page name %s\n", "ProtectedQuitPage");
            exitHyperDoc();
            return;
        }
        if (gWindow->fDownLinkStackIndex == MaxDownlinkDepth)
            fprintf(stderr, "exceeded maximum link nesting level\n");
        else
            gWindow->fDownLinkStack[gWindow->fDownLinkStackIndex++] =
                                                                gWindow->page;
        gWindow->page = page;
        displayPage(gWindow->page);
        gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;
    }
    else
        exitHyperDoc();
}
```

9.12.5 findPage

findPage takes as an argument the HyperDoc for a page name and returns the associated page.

⟨*hypertex*⟩+≡

```
static HyperDocPage *findPage(TextNode * node) {
    char *page_name;
    HyperDocPage *page;
    /* try and find the page name */
    page_name = printToString(node);
    page = (HyperDocPage *) hashFind(gWindow->fPageHashTable, page_name);
    if (page == NULL) {
        /* try to find the unknown page */
        page=(HyperDocPage *) hashFind(gWindow->fPageHashTable, "UnknownPage");
        if (page == NULL) {
            /* Yikes, Even that could not be found */
            fprintf(stderr, "Unknown page name %s\n", page_name);
        }
        else {
            if (page->type == UnloadedPageType)
                page->type = UlUnknownPage;
            else
                page->type = UnknownPage;
        }
    }
    return page;
}
```

9.12.6 downlink

Pushes a page onto the down link stack.

⟨*hypertex*⟩+≡

```
static void downlink(void) {
    if (gWindow->fDownLinkStackIndex == MaxDownlinkDepth)
       fprintf(stderr, "exceeded maximum link nesting level\n");
    else
       gWindow->fDownLinkStack[gWindow->fDownLinkStackIndex++] = gWindow->page;
}
```

9.12.7 memolink

⟨*hypertex*⟩+≡
```
static void memolink(void) {
    if (gWindow->fMemoStackIndex == MaxMemoDepth)
        fprintf(stderr, "exceeded maximum link nesting level\n");
    else {
        gWindow->fMemoStack[gWindow->fMemoStackIndex] = gWindow->page;
        gWindow->fDownLinkStackTop[gWindow->fMemoStackIndex++] =
                                                gWindow->fDownLinkStackIndex;
    }
}
```

9.12.8 killAxiomPage

⟨*hypertex*⟩+≡
```
static void killAxiomPage(HyperDocPage * page) {
    char command[512];
    sprintf(command, "(|htpDestroyPage| '%s)", page->name);
    sendLispCommand(command);
}
```

9.12.9 killPage

⟨*hypertex*⟩+≡
```
static void killPage(HyperDocPage * page) {
    page->scroll_off = 0;
    if (page->type == SpadGen) {
        hashDelete(gWindow->fPageHashTable, page->name);
        killAxiomPage(page);
        freePage(page);
    }
}
```

9.12.10 returnlink

Pops the memo stack.

⟨*hypertex*⟩+≡

```
static HyperDocPage *returnlink(void) {
    int i;
    if (gWindow->fMemoStackIndex == 0) {
        BeepAtTheUser();
        return NULL;
    }
    else {
        killPage(gWindow->page);
        for (i = gWindow->fDownLinkStackIndex - 1;
             i >= gWindow->fDownLinkStackTop[gWindow->fMemoStackIndex - 1];
             i--)
        {
            killPage(gWindow->fDownLinkStack[i]);
        }
        gWindow->fDownLinkStackIndex =
            gWindow->fDownLinkStackTop[--gWindow->fMemoStackIndex];
        return (gWindow->fMemoStack[gWindow->fMemoStackIndex]);
    }
}

/* pops a page if it can from the downlink stack */
```

9.12.11 uplink

⟨*hypertex*⟩+≡

```
static HyperDocPage *uplink(void) {
    if (gWindow->fDownLinkStackIndex == 0)
        return returnlink();
    else {
        killPage(gWindow->page);
        return (gWindow->fDownLinkStack[--gWindow->fDownLinkStackIndex]);
    }
}
```

9.12.12 windowlinkHandler

⟨*hypertex*⟩+≡

```
static void windowlinkHandler(TextNode * node) {
    char *page_name;
    /* first try and find the page */
    page_name = printToString(node);
    if (initTopWindow(page_name) == -1) {
        return;
    }
/*    gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;*/
}
```

9.12.13 makeWindowLink

⟨*hypertex*⟩+≡

```
void makeWindowLink(char *name) {
    if (initTopWindow(name) != -1)
{}/*        gWindow->fWindowHashTable = gWindow->page->fLinkHashTable; */
}
```

9.12.14 lispwindowlinkHandler

Since we are popping up a new window, then we had better change all the cursors right away. We won't get another chance at it.

⟨*hypertex*⟩+≡

```
static void lispwindowlinkHandler(HyperLink * link) {
    if (initTopWindow(NULL) != -1) {
        HyperDocPage *page = NULL;
        int frame = gWindow->fAxiomFrame;

        page = issueServerCommand(link);
        gWindow->fAxiomFrame = frame;
        gWindow->page = page;
/*        gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;*/
    }
}
```

9.12.15 pasteButton

⟨*hypertex*⟩+≡

```
static HyperDocPage *pasteButton(PasteNode * paste) {
    HyperDocPage *page = NULL;
    int pastewhere=paste->where;
    if ( paste->end_node ==NULL ||
         paste->begin_node==NULL ||
         paste->arg_node==NULL ) {
       BeepAtTheUser();
       return NULL;
    }
    page=parsePatch(paste);
/* paste has changed after this call so use pastewhere*/
    if (pastewhere && page ) {
        if (0 == strcmp(page->name, "ErrorPage"))
            page = NULL;
    }
    else
        BeepAtTheUser();
    return page;
}
```

9.12.16 helpForHyperDoc

⟨*hypertex*⟩+≡

```
void helpForHyperDoc(void) {
    HyperDocPage *page = NULL;
    /* do not do anything if we are already at the "no more help" page */
    if (0 == strcmp(gWindow->page->name, NoMoreHelpPage))
        return;
    /* if no help page recorded, use the standard "no more help" page */
    if (!gWindow->page->helppage)
        gWindow->page->helppage = allocString(NoMoreHelpPage);
    /* if we are on the main help page, use "no more help" page */
    if (0 == strcmp(gWindow->page->name, TopLevelHelpPage))
        gWindow->page->helppage = allocString(NoMoreHelpPage);
    page =
     (HyperDocPage *)hashFind(gWindow->fPageHashTable,gWindow->page->helppage);
    if (page)
        makeWindowLink(gWindow->page->helppage);
    else
        BeepAtTheUser();
}
```

9.12.17 findButtonInList

⟨*hypertex*⟩+≡

```
static HyperLink *findButtonInList(HDWindow * window, int x, int y) {
    ButtonList *bl;
    if (!window || window->page->type == UnloadedPageType)
        return NULL;
    for (bl = window->page->s_button_list; bl != NULL; bl = bl->next)
        if (x >= bl->x0 && x <= bl->x1 && y >= bl->y0 && y <= bl->y1)
            return bl->link;
    for (bl = window->page->button_list; bl != NULL; bl = bl->next)
        if (x >= bl->x0 && x <= bl->x1 && y >= bl->y0 && y <= bl->y1)
            return bl->link;
    return NULL;
}
```

9.12.18 getHyperLink

⟨*hypertex*⟩+≡

```
static HyperLink *getHyperLink(XButtonEvent * event) {
 HyperLink *l1, *l2;
 l1 =
   (HyperLink *)hashFind(gWindow->fWindowHashTable,(char *)&(event->window));
 if (l1)
   return l1;
 l2 = findButtonInList(gWindow, event->x, event->y);
 return l2;
}
```

9.12.19 handleButton

Handle a button pressed event. window is the subwindow in which the event occured, and button is the button which was pressed.

⟨*hypertex*⟩+≡

```
static void handleButton(int button, XButtonEvent * event) {
    HyperLink *link;
    HyperDocPage *page = NULL;
    char *page_name;

    /* handle mouse wheel (Gregory Vanuxem) */
    if (event->window == gWindow->fMainWindow ||
        event->window == gWindow->fScrollWindow) {
      if (button == 4) {
        scrollUp();
        return;
      }
      else if (button == 5) {
        scrollDown();
        return;
      }
    }

    /* find page name from sub-window handle */
    link = getHyperLink(event);
    if (link == NULL) {       /* user clicked on an inactive area */
/*      BeepAtTheUser();  */  /* I always thought this was annoying. RSS */
        return;
    }
    switch (link->type) {
      case Pastebutton:
        page = pasteButton(link->reference.paste);
        break;
      case Link:
        page_name = printToString(link->reference.node);
        page = (HyperDocPage *) hashFind(gWindow->fPageHashTable, page_name);
        break;
      case Helpbutton:
        helpForHyperDoc();
        page = NULL;
        break;
      case Scrollbar:
        scrollScroller(event);
        break;
      case Scrollupbutton:
```

```
        scrollUp();
        break;
      case Scrolldownbutton:
        scrollDown();
        break;
      case Inputstring:
        /* We must be changing input focus or getting a selection */
        changeInputFocus(link);
        if ( button == Button2 ) {
            XConvertSelection(gXDisplay, XA_PRIMARY, XA_STRING,
                XInternAtom(gXDisplay, "PASTE_SELECTION", False),
                gWindow->fMainWindow, CurrentTime);
            gSavedInputAreaLink = link;
        }
        break;
      case SimpleBox:
        page = NULL;
        toggleInputBox(link);
        break;
      case Radiobox:
        page = NULL;
        toggleRadioBox(link);
        break;
      case Quitbutton:
        quitHyperDoc();
        break;
      case Returnbutton:        /* pop memo information */
        page = returnlink();
        break;
      case Upbutton:            /* pop downlink information */
        page = uplink();
        break;
      case Downlink:
        page = findPage(link->reference.node);
        if (page  && NotSpecial(page->type))
            downlink();
        break;
      case Memolink:
        page = findPage(link->reference.node);
        if (page && NotSpecial(page->type))
            memolink();
        break;
      case Windowlink:
        page = findPage(link->reference.node);
        if (page && NotSpecial(page->type)) {
            windowlinkHandler(link->reference.node);
```

```
        gNeedIconName = 1;
        page = NULL;
    }
    break;
  case Lispwindowlink:
    lispwindowlinkHandler(link);
    gNeedIconName = 1;
    page = NULL;
    break;
  case LispMemoLink:
  case Spadmemolink:
    page = issueServerCommand(link);
    if (page && NotSpecial(page->type))
        memolink();
    break;
  case LispDownLink:
  case Spaddownlink:
    page = issueServerCommand(link);
    if (page && NotSpecial(page->type))
        downlink();
    break;
  case Spadlink:
  case Lisplink:
    page = issueServerCommand(link);
    break;
  case Lispcommand:
  case Qspadcall:
  case Spadcall:
    page = issueServerCommand(link);
    break;
  case Lispcommandquit:
  case Spadcallquit:
  case Qspadcallquit:
    page = issueServerCommand(link);
    exitHyperDoc();
    break;
  case Spadcommand:
  case Spadgraph:
  case Spadsrc:
    issueSpadcommand(gWindow->page, link->reference.node,
                     button == Button1, link->type);
    break;
  case Unixlink:
    page = issueUnixlink(link->reference.node);
    if (page && NotSpecial(page->type)) {
        downlink();
```

```
        }
        break;
      case Unixcommand:
        issueUnixcommand(link->reference.node);
        break;
      default:
        break;
    }
    if (page) {
        switch (page->type) {   /* check for special button types */
          case Quitbutton:
            exitHyperDoc();
            return;
          case Returnbutton:
            gWindow->page = returnlink();
            break;
          case Upbutton:
            gWindow->page = uplink();
            break;
          case ErrorPage:
          case UnknownPage:
          case UlUnknownPage:
            if (page->type == UlUnknownPage)
                page->type = UnloadedPageType;
            downlink();
            gWindow->page = page;
            break;
          default:              /* a normal link */
            gWindow->page = page;
            break;
        }
        if (link->type != Pastebutton)
            displayPage(gWindow->page);
              /* reset the window hash */
        gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;
    }
}
```

9.12.20 exitHyperDoc

⟨*hypertex*⟩+≡

```
void exitHyperDoc(void) {
    XEvent event;
    if (gSessionHashTable.num_entries == 1 || gParentWindow == gWindow) {
        freeHdWindow(gWindow);
        exit(0);
    }
    hashDelete(&gSessionHashTable, (char *)&gWindow->fMainWindow);
    /*
     * Now we quickly want to flush all the events associated with this
     * window from existence
     */
    XFlush(gXDisplay);
    while (XCheckWindowEvent(gXDisplay,gWindow->fMainWindow,
                             bigmask,&event)) { }
    while (XCheckWindowEvent(gXDisplay,gWindow->fScrollWindow,
                             bigmask,&event)) { }
    while (XCheckWindowEvent(gXDisplay,gWindow->fDisplayedWindow,
                             bigmask,&event)) { }
    while (XCheckWindowEvent(gXDisplay,gWindow->fScrollUpWindow,
                             bigmask,&event)) { }
    while (XCheckWindowEvent(gXDisplay,gWindow->fScrollDownWindow,
                             bigmask, &event)) { }
    while (XCheckWindowEvent(gXDisplay,gWindow->scrollbar,
                             bigmask,&event)) { }
    while (XCheckWindowEvent(gXDisplay,gWindow->scroller,
                             bigmask,&event)) { }
    XDestroyWindow(gXDisplay, gWindow->fMainWindow);
    freeHdWindow(gWindow);
    gWindow = NULL;
    gActiveWindow = -1;
    XFlush(gXDisplay);
}
```

9.12.21 setWindow

⟨*hypertex*⟩+≡

```
static int setWindow(Window window) {
    Window root, parent, *children, grandparent,myarg;
    HDWindow *htw;
    unsigned int nchildren;
    int st;
    myarg=window;
    nchildren = 0;
    htw = (HDWindow *) hashFind(&gSessionHashTable, (char *)&myarg);
    if (htw != NULL) {
        gWindow = htw;
        return 1;
    }
    st = XQueryTree(gXDisplay, myarg, &root, &parent, &children, &nchildren);
    if (st==0) goto ERROR;
    if (nchildren > 0)
        XFree(children);
    htw = (HDWindow *) hashFind(&gSessionHashTable, (char *)&parent);
    if (htw != NULL) {
        gWindow = htw;
        return 1;
    }
    else {
        /* check for a grandparent */
        st = XQueryTree(gXDisplay, parent, &root, &grandparent,
                        &children, &nchildren);
        if (st==0) goto ERROR;
        if (nchildren > 0)
            XFree(children);
        htw = (HDWindow *) hashFind(&gSessionHashTable, (char *)&grandparent);
        if (htw != NULL) {
            gWindow = htw;
            return 1;
        }
    }
    /*
     * fprintf(stderr, "window(%d) and it's parent(%d) aren't in
     * gSessionHashTable\n", window, parent);
     we never found that window. this happens if (not iff) we exit from
     an unfocused non-main window under certain wm's and click-to-type.
     the program returns here with the window handle that was just destroyed.
     So let's set the global gWindow to the main window.
     */
ERROR:
```

```
    gWindow=gParentWindow;
    return 0;
}

/*
 * This procedure whips thru the stack and clears all expose events for the
 * given routine
 */
```

9.12.22 clearExposures

⟨*hypertex*⟩+≡

```
static void clearExposures(Window w) {
    XEvent report;
    XFlush(gXDisplay);
    while (XCheckTypedWindowEvent(gXDisplay, w, Expose, &report));
}
```

9.12.23 getNewWindow

⟨*hypertex*⟩+≡

```
void getNewWindow(void) {
    int val;
    char buf[128];
    int frame;
    Window wid;
    HDWindow *htw;
    HyperDocPage *hpage;
    /*
     * If I am going to try and start a new window, then I should make sure I
     * have a coonection to listen on
     *
     * BUT This code is entered when a socket selects
     *
     * if (spadSocket == NULL) { spadSocket =
     * connect_to_local_server(SpadServer, MenuServer, 10); if (spadSocket
     * == NULL) { fprintf(stderr, "getNewWindow: Couldn't Connect to
     * SpadServer\n"); return -1; } }
     *
     */
    frame = get_int(spadSocket);
    val = get_int(spadSocket);
    switch (val) {
      case StartPage:
        initTopWindow(NULL);
        val = get_int(spadSocket);
        initScanner();
        inputType = FromSpadSocket;
        inputString = "";
        gWindow->page = parsePageFromSocket();
        gWindow->fAxiomFrame = frame;
        XFlush(gXDisplay);
        break;
      case LinkToPage:
        get_string_buf(spadSocket, buf, 128);
        if (initTopWindow(buf) == -1) {
            fprintf(stderr, "getNewWindow: Did not find page %s\n", buf);
            /* return -1; */
        }
        gWindow->fAxiomFrame = frame;
        break;
      case PopUpPage:
        val = get_int(spadSocket);
        initFormWindow(NULL, val);
```

```
    send_int(spadSocket, gWindow->fMainWindow);
    initScanner();
    inputType = FromSpadSocket;
    inputString = "";
    gWindow->page = parsePageFromSocket();
    computeFormPage(gWindow->page);
    XMapWindow(gXDisplay, gWindow->fMainWindow);
    gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;
    gWindow->fAxiomFrame = frame;
    XFlush(gXDisplay);
    break;
  case PopUpNamedPage:
    val = get_int(spadSocket);
    get_string_buf(spadSocket, buf, 128);

    if (initFormWindow(buf, val) == -1) {
        send_int(spadSocket, -1);
        break;
    }
    loadPage(gWindow->page);
    computeFormPage(gWindow->page);
    XMapWindow(gXDisplay, gWindow->fMainWindow);
    gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;
    gWindow->fAxiomFrame = frame;
    XFlush(gXDisplay);
    send_int(spadSocket, gWindow->fMainWindow);
    /* fprintf(stderr, "Window Id was %d\n", gWindow->fMainWindow); */
    break;
  case ReplaceNamedPage:
    wid = (Window) get_int(spadSocket);
    get_string_buf(spadSocket, buf, 128);
    htw = (HDWindow *) hashFind(&gSessionHashTable,(char *)&wid);
    if (htw == NULL) break;
    hpage = (HyperDocPage *) hashFind(gWindow->fPageHashTable, buf);
    if (hpage == NULL) break;
    gWindow = htw;
    gWindow->page = hpage;
    displayPage(gWindow->page);
    gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;
    clearExposures(gWindow->fMainWindow);
    clearExposures(gWindow->fScrollWindow);
    XFlush(gXDisplay);
    break;
  case ReplacePage:
    wid = (Window) get_int(spadSocket);
    setWindow(wid);
```

```
        initScanner();
        inputType = FromSpadSocket;
        inputString = "";
        gWindow->page = parsePageFromSocket();
        displayPage(gWindow->page);
        gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;
        clearExposures(gWindow->fMainWindow);
        clearExposures(gWindow->fScrollWindow);
        XFlush(gXDisplay);
        break;
      case KillPage:
        /* Here the user wishes to kill the page */
        wid = (Window) get_int(spadSocket);
        htw = (HDWindow *) hashFind(&gSessionHashTable,(char *)&wid);
        if (htw !=NULL) {
            gWindow = htw;
            exitHyperDoc();
            break;
          }
        break;
    }
  }
```

9.12.24 setCursor

⟨*hypertex*⟩+≡

```
static void setCursor(HDWindow *window,Cursor state) {
    if (state == gBusyCursor)
        XDefineCursor(gXDisplay, window->fMainWindow, gBusyCursor);
    else if (state == gActiveCursor)
        XDefineCursor(gXDisplay, window->fMainWindow, gActiveCursor);
    else
        XDefineCursor(gXDisplay, window->fMainWindow, gNormalCursor);
    XFlush(gXDisplay);
}
```

9.12.25 changeCursor

⟨*hypertex*⟩+≡

```
static void changeCursor(Cursor state, HDWindow *window) {
    if (window->fDisplayedCursor == state)
        return;
    window->fDisplayedCursor = state;
    setCursor(window, state);
}
```

9.12.26 handleMotionEvent

⟨*hypertex*⟩+≡

```
static void handleMotionEvent(XMotionEvent *event) {
    if (!gWindow)
        return;
    if (findButtonInList(gWindow, event->x, event->y) != NULL)
        changeCursor(gActiveCursor, gWindow);
    else
        changeCursor(gNormalCursor, gWindow);
}
```

9.12.27 initCursorState

⟨*hypertex*⟩+≡

```
static void initCursorState(HDWindow *window) {
    if (window) {
        int x, y, rx, ry, but;
        Window r, c;
        XQueryPointer(gXDisplay, window->fMainWindow,
                              &r, &c, &rx, &ry, &x, &y,(unsigned int *) &but);
        if (findButtonInList(window, x, y) != NULL)
            changeCursor(gActiveCursor, window);
        else
            changeCursor(gNormalCursor, window);
    }
}
```

9.12.28 initCursorStates

⟨*hypertex*⟩+≡
```
static void initCursorStates(void) {
    hashMap(&gSessionHashTable,(MappableFunction) initCursorState);
}
```

9.12.29 makeBusyCursor

⟨*hypertex*⟩+≡
```
static void makeBusyCursor(HDWindow *window) {
    changeCursor(gBusyCursor, window);
}
```

9.12.30 makeBusyCursors

⟨*hypertex*⟩+≡
```
static void makeBusyCursors(void) {
    hashMap(&gSessionHashTable, (MappableFunction)makeBusyCursor);
}
```

9.12.31 HyperDocErrorHandler

⟨*hypertex*⟩+≡
```
static int HyperDocErrorHandler(Display *display, XErrorEvent *xe) {
    if (xe->request_code != 15) {
        char buf[1024];
        XGetErrorText(display, xe->error_code, buf, sizeof(buf));
        fprintf(stderr, "error code = %d\n", xe->error_code);
        fprintf(stderr, "major op code = %d\n", xe->request_code);
        fprintf(stderr, "minor op code = %d\n", xe->minor_code);
        fprintf(stderr, "XID = %ld\n", xe->resourceid);
        fprintf(stderr, "%s\n", buf);
        if (xe->request_code != 15)
            exit(-1);
      }
    return(0);
}
```

9.12.32 setErrorHandlers

⟨*hypertex*⟩+≡

```
static void setErrorHandlers(void) {
    XSetErrorHandler(HyperDocErrorHandler);
}
```

9.13 Line Extent Computation

9.13.1 computeInputExtent

Computes the extent of the input string or box.

⟨*hypertex*⟩+≡

```
static void computeInputExtent(TextNode * node) {
    InputItem *item;
    int t_width;
    int num_lines;
    /* search the symbol table for the proper entry */
    item = node->link->reference.string;
    num_lines = item->num_lines;
    /*
     * Once we have gotten this far, we should just be able to calculate the
     * width using the normal font
     */
    t_width = (item->size + 1) * gInputFont->max_bounds.width + 10;
    if (gInLine)
        text_x += inter_word_space;
    if (text_x + t_width > right_margin) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    /* now figure out the height of the current window */
    node->height = line_height * (num_lines);
    node->y = text_y - line_height + node->height - 1;
    if (node->height > present_line_height)
        present_line_height = plh(node->height);
    node->width = t_width;
    gInLine = 1;
    text_x += t_width;
}
```

9.13.2 computePunctuationExtent

⟨*hypertex*⟩+≡

```
static void computePunctuationExtent(TextNode * node) {
    int twidth;
    int nextwidth;
    int incwidth;
    node->height = normal_textHeight;
    node->width = strlen(node->data.text);
    incwidth = twidth = XTextWidth(gTopOfGroupStack->cur_font, node->data.text,
                                   node->width);
    /* always check to see if there was some space in front of us */
    if (gInLine && (node->space & FRONTSPACE))
        twidth += inter_word_space;
    /*
     * now calcualte the width of the next one if it needs to be considered
     */
    if (!(node->space & BACKSPACE))
        nextwidth = totalWidth(node->next, Endtokens);
    else
        nextwidth = 0;
    if ((!(node->space & BACKSPACE)) &&
        (text_x + twidth + nextwidth > right_margin) && gInLine) {
        startNewline(present_line_height, node);
        if (gInAxiomCommand) {
            text_x = indent + spadcom_indent;
        }
        else
            text_x = indent;
    }
    if (node->space & FRONTSPACE)
        text_x += inter_word_space;
    node->x = text_x;
    /*
     * Now try to see if we should leave space after myself. Always leave
     * space when there is space
     */
    if (node->space & BACKSPACE) {
        switch (node->data.text[0]) {
          case '.':
          case '?':
          case '!':
            text_x += term_punct_space;
            break;
        }
    }
```

```
    text_x += incwidth;
    node->y = text_y - word_off_height;
    gInLine = 1;
}
```

9.13.3 computeWordExtent

⟨*hypertex*⟩+≡

```
static void computeWordExtent(TextNode * node) {
    int twidth;
    int nextwidth;
    int incwidth;
    node->height = normal_textHeight;
    node->width = strlen(node->data.text);
    incwidth = twidth = XTextWidth(gTopOfGroupStack->cur_font, node->data.text,
                                   node->width);
    /*
     * Now if we should drop some space in front of me, then add it to twidth
     */
    if (gInLine && node->space)
        twidth += inter_word_space;
    /*
     * Now what we should do is find all the things after us that have no
     * space in front and add there width on.
     */
    nextwidth = totalWidth(node->next, Endtokens);
    /*
     * Should we start a new line?
     */
    if (text_x + twidth + nextwidth > right_margin && gInLine) {
        startNewline(present_line_height, node);
        if (gInAxiomCommand) {
            text_x = indent + spadcom_indent;
        }
        else
            text_x = indent;
    }
    /*
     * Now see if we am on the beginning of a line, and if not add some space
     * if we need to
     */
    if (gInLine && node->space)
        text_x += inter_word_space;

    node->x = text_x;
    node->y = text_y - word_off_height;
    text_x += incwidth;
    gInLine = 1;
}
```

9.13.4 computeVerbatimExtent

⟨*hypertex*⟩+≡

```
static void computeVerbatimExtent(TextNode *node) {
    node->height = normal_textHeight;
    node->width = strlen(node->data.text);
    node->x = text_x;
    node->y = text_y - word_off_height;
    gInLine = 1;
    return;
}
```

9.13.5 computeSpadsrctxtExtent

⟨*hypertex*⟩+≡

```
static void computeSpadsrctxtExtent(TextNode *node) {
    node->height = normal_textHeight;
    node->width = strlen(node->data.text);
    if (gInLine) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    node->y = text_y - word_off_height;
    gInLine = 1;
    return;
}
```

9.13.6 computeDashExtent

⟨*hypertex*⟩+≡

```
static void computeDashExtent(TextNode *node) {
    int num_dashes;
    int twidth;
    int nextwidth;
    node->height = normal_textHeight;
    num_dashes = strlen(node->data.text);
    if (num_dashes > 1)
        twidth = node->width = num_dashes * dash_width;
    else
        twidth = node->width = XTextWidth(gTopOfGroupStack->cur_font,
                                          node->data.text, 1);
    if (gInLine && node->space)
        twidth += inter_word_space;
    /*
     * Now what we should do is find all the things after us that have no
     * space in front and add there width on.
     */
    nextwidth = totalWidth(node->next, Endtokens);
    /*
     * Should we start a new line?
     */
    if (text_x + twidth + nextwidth > right_margin) {
        startNewline(present_line_height, node);
        if (gInAxiomCommand) {
            text_x = indent + spadcom_indent;
        }
        else
            text_x = indent;
    }
    /*
     * Now see if we am on the beginning of a line, and if not add some space
     * if we need to
     */
    if (gInLine && node->space)
        text_x += inter_word_space;
    node->x = text_x;
    if (num_dashes > 1)
        node->y = text_y - dash_y;
    else
        node->y = text_y - word_off_height;
    text_x += node->width;
    gInLine = 1;
    return;
```

```
}
```

9.13.7 computeTextExtent

⟨*hypertex*⟩+≡

```
void computeTextExtent(TextNode *node) {
    for (; node != NULL; node = node->next) {
        switch (node->type) {
          case Endpastebutton:
            endpastebuttonExtent(node);
            break;
          case Paste:
            computePasteExtent(node);
            break;
          case Endpaste:
            if (gInLine) {
                startNewline(present_line_height, node);
                text_x = indent;
            }
            break;
          case Pastebutton:
            computePastebuttonExtent(node);
            break;
          case Ifcond:
            computeIfcondExtent(node);
            break;
          case Fi:
            break;
          case Endif:
            if (if_node == NULL) {
                return;
            }
            else
                endifExtent(node);
            break;
          case Endcenter:
            startNewline(present_line_height, node->next);
            popGroupStack();
            text_x = indent;
            break;
          case Pound:
          case Macro:
            /* check to see if we had space in front of me, if so add it */
            if (node->space && gInLine)
                text_x += inter_word_space;
            break;
          case Punctuation:
            computePunctuationExtent(node);
```

```
  break;
case Endmath:
  break;
case Endverbatim:
  if (gInLine) {
      startNewline(present_line_height, node);
      text_x = indent;
  }
  break;
case Spadsrctxt:
  computeSpadsrctxtExtent(node);
  break;
case Math:
  computeWordExtent(node);
  break;
case Verbatim:
  computeVerbatimExtent(node);
  break;
case WindowId:
case Word:
case Lsquarebrace:
case Rsquarebrace:
  computeWordExtent(node);
  break;
case Dash:
  computeDashExtent(node);
  break;
case HSpace:
  node->height = line_height;
  node->x = text_x;
  node->y = text_y;
  if (gInLine) {
   text_x +=
    (node->data.node != NULL ? atoi(node->data.node->data.text) : 1);
  }
  break;
case VSpace:
  node->height = line_height;
  node->x = text_x;
  node->y = text_y + present_line_height;;
  text_y +=
   (node->data.node != NULL ? atoi(node->data.node->data.text) : 1) +
      present_line_height;
  past_line_height = (node->data.node != NULL ?
                      atoi(node->data.node->data.text) : 1)
      + present_line_height;
```

```
        present_line_height = line_height;
        break;
      case Space:
        node->height = line_height;
        node->x = text_x;
        node->y = text_y;
        text_x += (gTopOfGroupStack->cur_font->max_bounds.width) *
          (node->data.node != NULL ? atoi(node->data.node->data.text) : 1);
        break;
      case Tab:
        node->height = line_height;
        text_x = indent + (gTopOfGroupStack->cur_font->max_bounds.width) *
          (node->data.node != NULL ? atoi(node->data.node->data.text) : 1);
        gInLine = 0;
        break;
      case Par:
        node->height = line_height;
        if (gInItem)
            text_x = indent;
        else
            text_x = indent + paragraph_space;
        if (gInLine) {
            startNewline(present_line_height, node);
        }
        break;
      case Newline:
        if (gInLine) {
            startNewline(present_line_height, node);
            text_x = indent;
        }
        break;
      case Horizontalline:
        if (gInLine) {
            startNewline(present_line_height, node);
            text_x = indent;
        }
        node->height = line_height;
        gInLine = 0;
        node->y = text_y - line_height / 2;
        node->x = text_x;
        startNewline(present_line_height, node);
        break;
      case Center:
        computeCenterExtent(node);
        break;
      case Box:
```

```
  computeBoxExtent(node);
  break;
case Mbox:
  computeMboxExtent(node);
  break;
case Beginitems:
case Begintitems:
  computeBeginItemsExtent(node);
  break;
case Enditems:
case Endtitems:
  popItemStack();
  if (gInLine) {
      startNewline(present_line_height, node);
  }
  text_x = indent;
  break;
case Titem:
  if (gInLine) {
      startNewline(present_line_height, node);
  }
  text_x = indent - item_space;
  break;
case Item:
  computeItemExtent(node);
  break;
case Mitem:
  computeMitemExtent(node);
  break;
case Upbutton:
case Returnbutton:
case Memolink:
case Downlink:
case Link:
case Windowlink:
  computeButtonExtent(node);
  break;
case Unixlink:
case Lisplink:
case Lispwindowlink:
case Spadcall:
case Spadcallquit:
case Qspadcall:
case Qspadcallquit:
case LispDownLink:
case LispMemoLink:
```

```
case Lispcommand:
case Lispcommandquit:
case Spadlink:
case Spaddownlink:
case Spadmemolink:
case Unixcommand:
  computeButtonExtent(node);
  break;
case Endbutton:
  endbuttonExtent(node);
  break;
case Endlink:
  if (link_node == NULL)
      return;
  else
      endbuttonExtent(node);
  break;
case Spadsrc:
  computeSpadsrcExtent(node);
  break;
case Spadcommand:
case Spadgraph:
  computeSpadcommandExtent(node);
  break;
case Endspadsrc:
  endSpadsrcExtent(node);
  break;
case Endspadcommand:
  endSpadcommandExtent(node);
  break;
case Indent:
  indent = left_margin +
      atoi(node->data.node->data.text) *
      (gTopOfGroupStack->cur_font->max_bounds.width);
  if (!gInLine)
      text_x = indent;
  break;
case Indentrel:
  indent += atoi(node->data.node->data.text) *
      (gTopOfGroupStack->cur_font->max_bounds.width);
  if (!gInLine)
      text_x = indent;
  break;
case Group:
  pushGroupStack();
  node->y = text_y;
```

```
  if (gInLine && node->space)
      text_x += inter_word_space;
  break;
case Endgroup:
  popGroupStack();
  break;
case Tableitem:
  pushGroupStack();
  node->y = text_y;
  if (gInLine && node->space)
      text_x += inter_word_space;
  break;
case Endtableitem:
  popGroupStack();
  return;
case Controlbitmap:
case Inputbitmap:
  if (node->width == -1)
      insertBitmapFile(node);
  computeImageExtent(node);
  break;
case Inputpixmap:
  if (node->width == -1)
      insertPixmapFile(node);
  computeImageExtent(node);
  break;
case Table:
  computeTableExtent(&node);
  break;
case BoldFace:
  computeBfExtent(node);
  break;
case Emphasize:
  computeEmExtent(node);
  break;
case It:
  computeItExtent(node);
  break;
case Rm:
case Sl:
case Tt:
  computeRmExtent(node);
  break;
case Inputstring:
  computeInputExtent(node);
  break;
```

```
            case SimpleBox:
            case Radiobox:
              computeIrExtent(node);
              break;
            case Endbox:
              text_x += box_width;
              break;
            case Endmacro:
            case Endparameter:
              break;
            case Description:
              bfTopGroup();
              break;
            case Enddescription:
              popGroupStack();
              if (gInDesc)
                  return;
              break;
            case Endscrolling:
              /*
               * What we should do here is if we am in the middle of a line, we
               * should end it here an now.
               */
              if (gInLine)
                  startNewline(present_line_height, node);
              break;
            case Noop:
              noop_count++;
              break;
            case Endinputbox:
            case Endheader:
            case Endtitle:
            case Endfooter:
            case Rbrace:
            case Free:
            case Bound:
            case Beep:
            case 0:
              break;
            default:
              fprintf(stderr, "computeTextExtent: Unknown node type %d\n",
                      node->type);
              break;
        }
    }
}
```

9.13.8 computeBeginItemsExtent

⟨*hypertex*⟩+≡

```
static void computeBeginItemsExtent(TextNode * node) {
    int store_x, store_y, lh;
    /*
     * This routine pushes the current item_stack, and then tries to set the
     * item_indent, and the indent level. It checks for an optional argument
     * to begin{items} and if found uses its width.
     */
    if (gInLine) {
        startNewline(present_line_height, node);
    }
    store_x = text_x, store_y = text_y, lh = present_line_height;
    text_x = indent;
    pushItemStack();
    gInItem++;
    item_indent = indent;
    if (node->data.node != NULL) {
        /* we have a desc */
        gInDesc = 1;
        computeTextExtent(node->data.node);
        gInDesc = 0;
        item_space = textWidth(node->data.node, Enddescription);
        text_x = store_x;
        text_y = store_y;
        present_line_height = lh;
        indent = item_indent + item_space;
    }
    else
        indent = item_indent + 30;
    gInLine = 0;
}
```

9.13.9 computeItemExtent

⟨*hypertex*⟩+≡

```
static void computeItemExtent(TextNode * node) {
    if (gInLine)
        startNewline(present_line_height, node);
    text_x = item_indent;
}
```

9.13.10 computeMitemExtent

⟨*hypertex*⟩+≡

```
static void computeMitemExtent(TextNode *node) {
    if (gInLine) {
        startNewline(present_line_height, node);
    }
    text_x = item_indent;
}
```

9.13.11 endifExtent

⟨*hypertex*⟩+≡

```
static void endifExtent(TextNode *node) {
    /*
     * This node has the responsibilty for updating text_x and text_y so that
     * they are the maxaimum width of teh else and then statements
     */
    text_x = if_node->x;
    text_y = if_node->y;
    if_node = NULL;
}
```

9.13.12 computeIfcondExtent

This routine checks the value of the condition and swaps in the `else` or the `then` depending.

⟨*hypertex*⟩+≡

```
static void computeIfcondExtent(TextNode *node) {
    TextNode *condnode = node->data.ifnode->cond;
    TextNode *tln = gLineNode;
    int store_x = text_x, store_y = text_y, lh = present_line_height;
    int then_x, then_y;
    /*
     * we have to compute the maximum width and height of the rest of the
     * text and stuff
     */
    pushGroupStack();
    if (gInLine && node->space)
        text_x += inter_word_space;
    computeTextExtent(node->data.ifnode->thennode);
    then_x = text_x;
    then_y = text_y;
    text_x = store_x;
    text_y = store_y;
    present_line_height = lh;
    gLineNode = tln;
    if (gInLine && node->space)
        text_x += inter_word_space;
    computeTextExtent(node->data.ifnode->elsenode);
    /* Now choose the best one that is biggest and put it into ifnode */
    if (then_y > text_y) {
        node->y = then_y;
        node->x = then_x;
    }
    else if (text_y > then_y) {
        node->y = text_y;
        node->x = text_x;
    }
    else if (text_x > then_x) {
        node->y = text_y;
        node->x = text_x;
    }
    else {
        node->y = then_y;
        node->x = then_x;
    }
    /* restore everything */
```

```
    text_x = store_x;
    text_y = store_y;
    present_line_height = lh;
    gLineNode = tln;
    node->width = 0;

    if_node = node;
    if (gInLine && node->space)
        text_x += inter_word_space;
    if (checkCondition(condnode)) {
        node->next = node->data.ifnode->thennode;
    }
    else {
        node->next = node->data.ifnode->elsenode;
    }
    popGroupStack();
}
```

9.13.13 computeCenterExtent

⟨*hypertex*⟩+≡

```
static void computeCenterExtent(TextNode * node) {
    if (gInLine)
        startNewline(present_line_height, node);
    centerTopGroup();
    if (gLineNode)
        text_x = indent;
    else {
        fprintf(stderr, "(HyperDoc) Internal error: unexpected state ");
        fprintf(stderr, "in computeCenterExtent.\n");
        exit(-1);
    }
}
```

9.13.14 computeBfExtent

⟨*hypertex*⟩+≡

```
static void computeBfExtent(TextNode *node) {
    if (gInLine && node->space)
        text_x += inter_word_space;
    node->x = text_x;
    node->y = text_y;
    bfTopGroup();
}
```

9.13.15 computeEmExtent

⟨*hypertex*⟩+≡

```
static void computeEmExtent(TextNode *node) {
    if (gInLine && node->space)
        text_x += inter_word_space;
    node->x = text_x;
    node->y = text_y;
    if (gTopOfGroupStack->cur_font == gEmFont)
        rmTopGroup();
    else
        emTopGroup();
}
```

9.13.16 computeItExtent

⟨*hypertex*⟩+≡

```
static void computeItExtent(TextNode *node) {
    if (gInLine && node->space)
        text_x += inter_word_space;
    node->x = text_x;
    node->y = text_y;
}
```

9.13.17 computeRmExtent

⟨*hypertex*⟩+≡

```
static void computeRmExtent(TextNode *node) {
    if (gInLine && node->space)
        text_x += inter_word_space;
    node->x = text_x;
    node->y = text_y;
    rmTopGroup();
}
```

9.13.18 computeButtonExtent

⟨*hypertex*⟩+≡

```
static void computeButtonExtent(TextNode *node) {
    int twidth;
    /*int store_x = text_x;*/
    /*int store_y = text_y;*/
    /*int lh = present_line_height;*/
    pushActiveGroup();
    /* First see if we should leave a little space in front of myself * */
    if (gInLine && node->space)
        text_x += inter_word_space;

    twidth = textWidth(node->next, Endbutton);
    if (gInLine && node->space)
        text_x += inter_word_space;
    if (text_x + twidth > right_margin && gInLine) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    node->y = text_y;
    link_node = node;
}
```

9.13.19 endbuttonExtent

⟨*hypertex*⟩+≡

```
static void endbuttonExtent(TextNode *node) {
    int temp;
    int height;
    int twidth;
    int y;
    int maxx;
    maxx = maxX(link_node, Endbutton);
    link_node->width = twidth = textWidth(link_node->next, Endbutton);
    height = link_node->y;
    temp = textHeight(link_node->next, Endbutton);
    link_node->height = temp - link_node->y + line_height;
    if (gInLine)
        y = text_y;
    else
        y = text_y - past_line_height;
    if (y > height) {
        link_node->y = temp;    /* height + link_node->height -
                                 * normal_textHeight; */
        link_node->width = maxx - indent;
        if (gInLine) {
            startNewline(present_line_height, node);
            text_x = indent;
        }
    }
    else {
        link_node->width = twidth;
        link_node->y = text_y + link_node->height - line_height;
    }
    popGroupStack();
    link_node = NULL;
}
```

9.13.20 computePastebuttonExtent

⟨*hypertex*⟩+≡

```
static void computePastebuttonExtent(TextNode *node) {
    int twidth;
    pushActiveGroup();
    /* First see if we should leave a little space in front of myself * */
    if (gInLine && node->space)
        text_x += inter_word_space;
    twidth = textWidth(node->next, Endpastebutton);
    if (gInLine && node->space)
        text_x += inter_word_space;
    if (text_x + twidth > right_margin && gInLine) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    node->y = text_y;
    paste_node = node;
    return;
}
```

9.13.21 endpastebuttonExtent

⟨*hypertex*⟩+≡

```
static void endpastebuttonExtent(TextNode *node) {
    int temp;
    int height;
    int twidth;
    paste_node->width = twidth = textWidth(paste_node->next, Endpastebutton);
    height = paste_node->y;
    temp = textHeight(paste_node->next, Endpastebutton);
    paste_node->height = temp - paste_node->y + line_height;
    if (text_y > height) {
        paste_node->y = temp;
        paste_node->width = right_margin - indent;
        if (gInLine) {
            startNewline(present_line_height, node);
            text_x = indent;
        }
    }
    else {
        paste_node->width = twidth;
        paste_node->y = text_y + paste_node->height - line_height;
    }
    popGroupStack();
    paste_node = NULL;
    gInLine = 1;
}
```

9.13.22 computePasteExtent

⟨*hypertex*⟩+≡

```
static void computePasteExtent(TextNode *node) {
    if (gInLine) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    node->y = text_y;
    node->height = line_height;
}
```

9.13.23 computeSpadcommandExtent

Compute the text extent of a spadcommand node.

⟨*hypertex*⟩+≡

```
static void computeSpadcommandExtent(TextNode *node) {
    /*
     * From now on if there is an example which will take over a line, then
     * it will start and end with a newline
     */
    /*int height;*/
    int t_width;
    /*int store_x = text_x;*/
    /*int store_y = text_y;*/
    /*int lh = present_line_height;*/
    gInAxiomCommand = 1;
    pushSpadGroup();
    /* Check to see if we should space in front of myself         */
    if (gInLine && node->space)
        text_x += inter_word_space;
    t_width = textWidth(node->next, Endspadcommand);
    if (gInLine && ((text_x + t_width) > right_margin)) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    node->y = text_y;
    spad_node = node;
}
```

9.13.24 computeSpadsrcExtent

⟨*hypertex*⟩+≡

```
static void computeSpadsrcExtent(TextNode *node) {
    /*
     * From now on if there is an example which will take over a line, then
     * it will start and end with a newline
     */
    /*int store_x = text_x;*/
    /*int store_y = text_y;*/
    /*int lh = present_line_height;*/
    gInAxiomCommand = 1;
    pushSpadGroup();
    if (gInLine) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    node->y = text_y;
    spad_node = node;
}
```

9.13.25 endSpadcommandExtent

⟨*hypertex*⟩+≡

```
static void endSpadcommandExtent(TextNode *node) {
    int temp;
    int height;
    int twidth;
    int maxx;
    /*int y = (gInLine) ? (text_y) : (text_y - past_line_height);*/
    maxx = maxX(spad_node, Endspadcommand);
    twidth = spad_node->width = textWidth(spad_node->next, Endspadcommand);
    height = spad_node->y;
    temp = textHeight(spad_node->next, Endspadcommand);
    spad_node->height = temp - height + line_height;
    if (text_y > height && gInLine) {
        spad_node->y = temp;
        spad_node->width = maxx - indent;
        startNewline(present_line_height, node);
        text_x = indent;
    }
    else {
        spad_node->width = twidth;
        spad_node->y = text_y - line_height + spad_node->height;
    }
    popGroupStack();
    gInAxiomCommand = 0;
    spad_node = NULL;
}
```

9.13.26 endSpadsrcExtent

⟨*hypertex*⟩+≡

```
static void endSpadsrcExtent(TextNode *node) {
    int temp;
    int height;
    int twidth;
    int maxx;
    int y = (gInLine) ? (text_y) : (text_y - past_line_height);
    maxx = maxX(spad_node, Endspadsrc);
    twidth = spad_node->width = textWidth(spad_node->next, Endspadsrc);
    height = spad_node->y;
    temp = textHeight(spad_node->next, Endspadsrc);
    spad_node->height = temp - height + line_height;
    if (y > height && gInLine) {
        spad_node->y = temp;
        spad_node->width = maxx - indent;
        startNewline(present_line_height, node);
        text_x = indent;
    }
    else {
        spad_node->width = twidth;
        spad_node->y = text_y - line_height + spad_node->height;
    }
    popGroupStack();
    gInAxiomCommand = 0;
    spad_node = NULL;
}
```

9.13.27 computeMboxExtent

⟨*hypertex*⟩+≡

```
static void computeMboxExtent(TextNode *node) {
    node->width = textWidth(node->next, Endmbox);
    if (node->space)
        text_x += inter_word_space;
    if (text_x + node->width > right_margin) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    node->y = text_y;
}
```

9.13.28 computeBoxExtent

⟨*hypertex*⟩+≡

```
static void computeBoxExtent(TextNode *node) {
    int t_width;
    /*
     * First thing we do is see if we need to skip some space in front of the
     * word
     */
    if (gInLine && node->space)
        text_x += inter_word_space;
    /* Calculate the actual width of the box */
    t_width = textWidth(node->next, Endbox) + 2 * box_width;
    if (text_x + t_width > right_margin) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    text_x = text_x + box_width;
    node->y = text_y - 2;
    node->width = t_width;
    node->height = line_height - 2;
    gInLine = 1;
}
```

9.13.29 computeIrExtent

⟨*hypertex*⟩+≡

```
static void computeIrExtent(TextNode *node) {
    int t_width;
    /*
     * First thing we do is see if we need to skip some space in front of the
     * word
     */
    if (gInLine && node->space)
        text_x += inter_word_space;
    /* Calculate the actual width of the box */
    t_width = node->width;
    if (text_x + t_width > right_margin) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    if (node->height > line_height) {
        node->height = present_line_height
                     = plh(node->height + inter_line_space);
        node->y = text_y + node->height - normal_textHeight;
    }
    else {
        node->y = text_y - line_height + node->height;
    }
    gInLine = 1;
    text_x += node->width;
}
```

9.13.30 computeImageExtent

Read a bitmap file into memory.

⟨*hypertex*⟩+≡

```
static void computeImageExtent(TextNode *node) {
    if (text_x + node->width > right_margin) {
        startNewline(present_line_height, node);
        text_x = indent;
    }
    node->x = text_x;
    if (node->height > line_height) {
        present_line_height = plh(node->height + inter_line_space);
        node->y = text_y + node->height - line_height;
    }
    else {
        node->y = text_y - line_height + node->height;
    }
    text_x += node->width;
    gInLine = 1;
}
```

9.13.31 computeTableExtent

Compute the coordinates of the entries in a table.

⟨*hypertex*⟩+≡

```
static void computeTableExtent(TextNode **node) {
    int num_cols, num_lines;
    int max_width = 0, node_width, col_width;
    int x, y, num_entries = 0,/* n=0, */ screen_width, table_top;
    TextNode *front = *node;
    TextNode *tn;
    gInTable = 1;
    front->x = text_x;
    front->y = text_y;
    for (tn=front->next; tn->type != Endtable; num_entries++, tn = tn->next) {
        /* Now we need to scan the table group by group */
        node_width = textWidth(tn->next, Endtableitem);
        if (node_width > max_width)
            max_width = node_width;
        /* Get to the beginning og the next group */
        for (; tn->type != Endtableitem; tn = tn->next);
    }
    col_width = max_width + min_inter_column_space;
    screen_width = gWindow->width - right_margin_space - indent;
    num_cols = screen_width / col_width;
    if (num_cols == 0)
        num_cols = 1;
    num_lines = num_entries / num_cols;
    if (num_entries % num_cols != 0)
        ++num_lines;
    if (gInLine) {
        startNewline(present_line_height, *node);
    }
    table_top = text_y;
    num_cols = num_entries / num_lines;
    if (num_entries % num_lines != 0)
        ++num_cols;
    col_width = screen_width / num_cols;
    for (tn = front->next, x = 0; x < num_cols; x++)
        for (y = 0; y < num_lines && tn->type != Endtable; y++) {
            if (num_cols == 1 && y > 0)
                text_y += line_height;
            else
                text_y = table_top + y * line_height;
            text_x = indent + x * col_width;
            gInLine = 0;
```

```
            computeTextExtent(tn->next);
            for (; tn->type != Endtableitem; tn = tn->next);
            tn = tn->next;
        }
    front->height = num_lines * line_height;
    front->width = screen_width;
    text_x = indent;
    if (num_cols == 1)
        text_y += line_height;
    else
        text_y = table_top + front->height;
    *node = tn;
    gInLine = 0;
}
```

9.13.32 computeTitleExtent

⟨*hypertex*⟩+≡

```
void computeTitleExtent(HyperDocPage *page) {
    right_margin_space = non_scroll_right_margin_space;
    page->title->height = twheight + gWindow->border_width;
    page->title->x = gWindow->border_width +
                     2 * twwidth + (int) gWindow->border_width / 2;
    gLineNode = page->title->next;
    initTitleExtents(page);
    text_y = top_margin + line_height;
    computeTextExtent(page->title->next);
    page->title->height = max(textHeight(page->title->next, Endtitle),
                              twheight);
}
```

9.13.33 computeHeaderExtent

⟨*hypertex*⟩+≡

```
void computeHeaderExtent(HyperDocPage *page) {
    /*
     * Hopefully we will soon be able to actually compute the needed height
     * for the header here
     */
    int ty; /* UNUSED */
    gExtentRegion = Header;
    right_margin_space = non_scroll_right_margin_space;
    initExtents();
    ty = text_y = 3 * top_margin +
                  line_height + max(page->title->height, twheight);
    gLineNode = page->header->next;
    computeTextExtent(page->header->next);
    page->header->height = textHeight(page->header->next, Endheader);
    if (page->header->height) {
        page->header->height += 1 / 2 * line_height;
        page->top_scroll_margin = (gInLine) ? text_y : text_y-past_line_height;
        if (!(page->pageFlags & NOLINES))
            page->top_scroll_margin += (int) line_height / 2;
        page->top_scroll_margin += gWindow->border_width + 2 * top_margin;
    }
    else {
        page->top_scroll_margin = page->title->height + gWindow->border_width +
            2 * scroll_top_margin;
    }
}
```

9.13.34 computeFooterExtent

⟨*hypertex*⟩+≡

```
void computeFooterExtent(HyperDocPage * page) {
    if (page->footer) {
        gExtentRegion = Footer;
        right_margin_space = non_scroll_right_margin_space;
        initExtents();
        present_line_height = line_height;
        text_y = line_height;
        gLineNode = page->footer->next;
        computeTextExtent(page->footer->next);
        page->footer->height = textHeight(page->footer->next, Endfooter);
        if (page->footer->height) {
            if ((!page->pageFlags & NOLINES))
                page->footer->height += (int) line_height / 2;
            page->bot_scroll_margin = gWindow->height -
                page->footer->height - bottom_margin
                - gWindow->border_width + top_margin;
        }
        else
            page->bot_scroll_margin = gWindow->height;
    }
}
```

9.13.35 computeScrollingExtent

⟨*hypertex*⟩+≡

```
void computeScrollingExtent(HyperDocPage *page) {
    /* Check to see if there is a scrolling region  */
    if (!page->scrolling) {
        return;
    }
    noop_count = 0;
    /* If there is then compute all the proper locations */
    gExtentRegion = Scrolling;
    right_margin_space = non_scroll_right_margin_space + gScrollbarWidth;
    initExtents();
    text_y = line_height;
    gLineNode = page->scrolling->next;
    computeTextExtent(page->scrolling->next);
    /*
     * the following is an attempt to fix the bug where one cannot scroll
     * down to a bitmap that is opened at the bottom of a page.
     */
    /*
     * TTT trial if(!gInLine)
     */
    if (0) {
        text_y = text_y - past_line_height;
    }
    else if (present_line_height > line_height)
        text_y = text_y + present_line_height - line_height;
    page->scrolling->height = text_y;
}
```

9.13.36 startNewline

The startNewline function updates the current header node, and also allocates if needed memory for the next Line Header. It also assigns the first TextNode on the line to the structure, because this is the last time I will be able to do this.

⟨*hypertex*⟩+≡

```
void startNewline(int distance, TextNode * node) {
    if (gLineNode != NULL) {
        if (gTopOfGroupStack->center)
            centerNodes(gLineNode, node);
        gLineNode = node;
    }
    text_y += distance;
    past_line_height = distance;
    present_line_height = line_height;
    gInLine = 0;
}
```

9.13.37 centerNodes

The centerNodes goes through and centers all the text between the two given nodes.

⟨*hypertex*⟩+≡

```
static void centerNodes(TextNode * begin_node, TextNode * end_node) {
    int begin_x, end_x, wmid_x, offset, mid_x;
    TextNode *node;
    end_x = text_x;
    begin_x = Xvalue(begin_node);
    mid_x = (int) (end_x + begin_x) / 2;
    wmid_x = (int) (right_margin + indent) / 2;
    if (mid_x > wmid_x)
        offset = 0;
    else
        offset = wmid_x - mid_x;
    for (node = begin_node; node != end_node; node = node->next)
        if (node->x > 0)
            node->x += offset;
}
```

9.13.38 punctuationWidth

⟨*hypertex*⟩+≡

```
static int punctuationWidth(TextNode * node) {
    int twidth, width = strlen(node->data.text);
    twidth = XTextWidth(gTopOfGroupStack->cur_font, node->data.text, width);
    /* check to see if there was some space in front */
    if (gInLine && (node->space & FRONTSPACE))
        twidth += inter_word_space;
    return twidth;
}
```

9.13.39 inputStringWidth

⟨*hypertex*⟩+≡

```
static int inputStringWidth(TextNode * node) {
    InputItem *item;
    int t_width;
    /** search the symbol table for the proper entry **/
    item = node->link->reference.string;
    /** Once I have gotten this far, I should just be able to calculate
      the width using the normal font **/
    t_width = (item->size + 1) * gInputFont->max_bounds.width + 10;
    return t_width;
}
```

9.13.40 wordWidth

⟨*hypertex*⟩+≡

```
static int wordWidth(TextNode * node) {
    int twidth, len = strlen(node->data.text);
    twidth = XTextWidth(gTopOfGroupStack->cur_font, node->data.text, len);
    if (node->space & FRONTSPACE)
        twidth += inter_word_space;
    return twidth;
}
```

9.13.41 verbatimWidth

⟨*hypertex*⟩+≡

```
static int verbatimWidth(TextNode * node) {
    int twidth, len = strlen(node->data.text);
    twidth = XTextWidth(gTopOfGroupStack->cur_font, node->data.text, len);
    if (node->space)
        twidth += inter_word_space;
    return twidth;
}
```

9.13.42 widthOfDash

⟨*hypertex*⟩+≡

```
static int widthOfDash(TextNode * node) {
    int num_dashes, twidth;
    num_dashes = strlen(node->data.text);
    if (num_dashes > 1)
        twidth = node->width = num_dashes * dash_width;
    else
        twidth = node->width = XTextWidth(gTopOfGroupStack->cur_font,
                                          node->data.text, 1);
    if (node->space)
        twidth += inter_word_space;
    return twidth;
}
```

9.13.43 textWidth

Return the gWindow-¿width in pixels of the given text node, when displayed

⟨*hypertex*⟩+≡

```
int textWidth(TextNode * node, int Ender) {
    int twidth = 0, num_words;
    for (num_words = 0; node != NULL; num_words++, node = node->next) {
        if (Ender == Endtokens) {
            if (node->type == Endtokens)
                return twidth;
        }
        else if (node->type == Ender)
            return twidth;
        switch (node->type) {
          case Macro:
          case Pound:
            if (node->space && gInLine)
                twidth += inter_word_space;
            break;
          case Punctuation:
            twidth += punctuationWidth(node);
            break;
          case Dash:
            if (gInLine && node->space)
                twidth += inter_word_space;
            twidth += widthOfDash(node);
            break;
          case Verbatim:
          case Spadsrctxt:
            twidth += verbatimWidth(node);
            break;
          case Lsquarebrace:
          case Rsquarebrace:
          case Word:
            twidth += wordWidth(node);
            break;
          case Box:
            twidth += 2 * box_space;
            break;
          case Link:
          case Downlink:
          case Memolink:
          case Windowlink:
          case LispMemoLink:
          case Lispwindowlink:
```

```
case Lisplink:
case Unixlink:
case Spadcall:
case Spadcallquit:
case Qspadcall:
case Qspadcallquit:
case LispDownLink:
case Lispcommand:
case Lispcommandquit:
case Spadlink:
case Spaddownlink:
case Spadmemolink:
case Unixcommand:
case Upbutton:
case Returnbutton:
case Description:
  pushActiveGroup();
  break;
case Endbutton:
case Endspadcommand:
case Enddescription:
  popGroupStack();
  break;
case Endlink:
  popGroupStack();
  break;
case Inputstring:
  twidth += inputStringWidth(node);
  break;
case SimpleBox:
case Radiobox:
  twidth += node->width + ((node->space) ? inter_word_space : 0);
  break;
case Spadcommand:
case Spadgraph:
  pushSpadGroup();
  break;
case VSpace:
  break;
case HSpace:
  twidth +=
    (node->data.node != NULL ? atoi(node->data.node->data.text) : 1);
  break;
case Space:
  twidth += (gTopOfGroupStack->cur_font->max_bounds.width) *
    (node->data.node != NULL ? atoi(node->data.node->data.text) : 1);
```

```
  break;
case Tab:
  twidth = (gTopOfGroupStack->cur_font->max_bounds.width) *
    (node->data.node != NULL ? atoi(node->data.node->data.text) : 1);
  break;
case Table:
  twidth = gWindow->width - left_margin - right_margin_space;
  break;
case Tableitem:
case Group:
  twidth += (node->space) ? inter_word_space : 0;
  pushGroupStack();
  break;
case BoldFace:
  if (node->space)
      twidth += inter_word_space;
  bfTopGroup();
  break;
case Emphasize:
  if (node->space)
      twidth += inter_word_space;
  if (gTopOfGroupStack->cur_font == gRmFont)
      emTopGroup();
  else
      rmTopGroup();
  break;
case It:
  if (node->space)
      twidth += inter_word_space;
  emTopGroup();
  break;
case Rm:
case Sl:
case Tt:
  if (node->space)
      twidth += inter_word_space;
  rmTopGroup();
  break;
case Endgroup:
  popGroupStack();
  break;
case Controlbitmap:
case Inputbitmap:
  if (node->width == -1)
      insertBitmapFile(node);
  twidth += node->width;
```

```
            break;
          case Inputpixmap:
            if (node->width == -1)
                insertPixmapFile(node);
            twidth += node->width;
            break;
          case Mbox:
          case Indent:
          case Endmacro:
          case Free:
          case Bound:
          case Beep:
          case Item:
          case Titem:
          case Beginitems:
          case Noop:
          case Endinputbox:
          case Fi:
          case Ifcond:
          case Endif:
          case Begintitems:
          case Enditems:
          case Endtitems:
          case Endtableitem:
          case Endtable:
          case Endparameter:
          case Endbox:
          case Endheader:
          case Endfooter:
          case Endscrolling:
          case Endverbatim:
          case Endspadsrc:
            break;
          case Newline:
            /* WOw, I guess I should ertunr a really big number */
            twidth += gWindow->width;
            break;
          default:
            /*
             * fprintf(stderr, "Unknown nodetype %d in textWidth\n",
             * node->type);
             */
            break;
        }
    }
    return twidth;
```

```
}

/*
 */
```

9.13.44 totalWidth

The totalWidth function traces through the nodes, until it finds a blank space. It is used by computeWordExtent, and computePunctuation extent to determine. How far we go before we actually see white space.

⟨*hypertex*⟩+≡

```
int totalWidth(TextNode * node, int Ender) {
    int twidth = 0;
    for (; (node != NULL); node = node->next) {
        if (Ender == Endtokens) {
            if (node->type >= Endtokens)
                return twidth;
        }
        else if (node->type == Ender)
            return twidth;
        /*
         * The first thing we check for is to see if there was space in front
         * of the current node, if so we are done
         */
        if (node->space)
            return twidth;
        /*** Else depending on the node type ***/
        switch (node->type) {
          case Noop:
          case Endinputbox:
          case Pound:
          case Ifcond:
          case Fi:
          case Endif:
            break;
          case Rsquarebrace:
          case Punctuation:
          case Word:
          case Dash:
            twidth += XTextWidth(gTopOfGroupStack->cur_font, node->data.text,
                                 strlen(node->data.text));
            break;
          case Box:
          case Link:
          case Downlink:
          case Memolink:
          case Windowlink:
          case LispMemoLink:
          case Lispwindowlink:
          case Lisplink:
```

```
        case Unixlink:
        case Spadcall:
        case Spadcallquit:
        case Qspadcall:
        case Qspadcallquit:
        case LispDownLink:
        case Lispcommand:
        case Lispcommandquit:
        case Spadlink:
        case Spaddownlink:
        case Spadmemolink:
        case Unixcommand:
        case Inputstring:
        case SimpleBox:
        case Radiobox:
        case Upbutton:
        case Returnbutton:
        case Spadcommand:
        case Spadgraph:
        case VSpace:
        case HSpace:
        case Space:
        case Table:
        case Group:
        case Controlbitmap:
        case Inputbitmap:
        case Inputpixmap:
        case Free:
        case Beep:
        case Bound:
        case Lsquarebrace:
        case BoldFace:
        case Emphasize:
        case It:
        case Rm:
        case Sl:
        case Tt:
        case Newline:
        case Verbatim:
        case Spadsrctxt:
          return twidth;
        default:
          break;
      }
  }
  return twidth;
```

```
}
```

9.13.45 initExtents

The initExtents function initialize some text size variables

⟨*hypertex*⟩+≡

```
void initExtents(void) {
    present_line_height = line_height;
    gInLine = 0;
    gInItem = 0;
    gInAxiomCommand = 0;
    item_indent = 0;
    gInDesc = 0;
    indent = left_margin;
    text_x = indent;
    gTopOfGroupStack->cur_font = gRmFont;
    gTopOfGroupStack->cur_color = gRmColor;
    right_margin = gWindow->width - right_margin_space;
    clearItemStack();
}
```

9.13.46 initTitleExtents

The initTitleExtents function initialize some title text size variables.

⟨*hypertex*⟩+≡

```
void initTitleExtents(HyperDocPage * page) {
    present_line_height = line_height;
    gInLine = 0;
    gInAxiomCommand = 0;
    item_indent = 0;
    gInDesc = 0;
    indent = left_margin + page->title->x;
    text_x = indent;
    gTopOfGroupStack->cur_font = gRmFont;
    gTopOfGroupStack->cur_color = gRmColor;
    right_margin = gWindow->width - right_margin_space -
                   gWindow->border_width - 2 * twwidth;
    clearItemStack();
}
```

9.13.47 initText

The initText function initialize some text size variables.

⟨*hypertex*⟩+≡
```
void initText(void) {
    normal_textHeight = gRmFont->ascent + gRmFont->descent;
    line_height = gRmFont->ascent + gRmFont->descent + inter_line_space;
    word_off_height = line_height - normal_textHeight;
    space_width = gRmFont->max_bounds.width;
}
```

9.13.48 textHeight

The textHeight function returns the height of a piece of formatted text in pixels.

⟨*hypertex*⟩+≡
```
int textHeight(TextNode * node, int Ender) {
    cur_height = 0;
    return textHeight1(node, Ender);
}
```

9.13.49 textHeight1

The textHeight1 function is the recursive part of textHeight.

⟨*hypertex*⟩+≡

```
static int textHeight1(TextNode * node, int Ender) {
    for (; node != NULL; node = node->next) {
        if (Ender == Endtokens) {
            if (node->type > -Endtokens)
                return cur_height;
        }
        else if (node->type == Ender)
            return cur_height;
        switch (node->type) {
          case Center:
          case Downlink:
          case Link:
          case Spadcommand:
          case Spadgraph:
          case Upbutton:
          case Returnbutton:
          case Windowlink:
          case Memolink:
          case Lispwindowlink:
          case Lisplink:
          case Unixlink:
          case Spadcall:
          case Spadcallquit:
          case Qspadcall:
          case Qspadcallquit:
          case LispDownLink:
          case LispMemoLink:
          case Lispcommand:
          case Lispcommandquit:
          case Spadlink:
          case Spaddownlink:
          case Spadmemolink:
          case Unixcommand:
          case SimpleBox:
          case Radiobox:
          case Group:
          case Box:
          case Controlbitmap:
          case Inputbitmap:
          case Inputpixmap:
          case Horizontalline:
```

```
case Punctuation:
case Lsquarebrace:
case Rsquarebrace:
case Word:
case Verbatim:
case Math:
case Spadsrctxt:
case Dash:
case Inputstring:
  cur_height = max(node->y, cur_height);
  break;
case Mbox:
case Macro:
case Pound:
case Emphasize:
case BoldFace:
case It:
case Rm:
case Sl:
case Tt:
case Endparameter:
case Description:
case Enddescription:
case Noop:
case Fi:
case Ifcond:
case Endif:
case Endinputbox:
case Tab:
case Newline:
case Space:
case VSpace:
case HSpace:
case Beginitems:
case Begintitems:
case Endtitems:
case Titem:
case Enditems:
case Endtable:
case Endtableitem:
case Item:
case Par:
case Beep:
case Free:
case Bound:
case Endgroup:
```

```
            case Endcenter:
            case Endbutton:
            case Endmacro:
            case Tableitem:
            case Endlink:
            case Endspadcommand:
            case Indent:
            case Indentrel:
            case Endbox:
            case Endmbox:
            case Table:
            case Endverbatim:
            case Endmath:
            case Spadsrc:
            case Endspadsrc:
              break;
            case Beginscroll:
            case Endscroll:
              break;
            case Endscrolling:
              return cur_height;
            default:
              /*
               * fprintf(stderr, "textHeight1: Unknown Node Type %d\n",
               * node->type);
               */
              break;
          }
      }
      return cur_height;
  }
```

9.13.50 maxX

The maxX function returns the height of a piece of formatted text in pixels.

⟨*hypertex*⟩+≡

```
int maxX(TextNode * node, int Ender) {
    maxXvalue = 0;
    for (; node != NULL; node = node->next) {
        if (Ender == Endtokens) {
            if (node->type >= Endtokens)
                return maxXvalue;
        }
        else if (node->type == Ender)
            return maxXvalue;
        switch (node->type) {
          case Lsquarebrace:
          case Rsquarebrace:
          case Word:
            maxXvalue = max(maxXvalue, node->x + wordWidth(node));
            break;
          case Verbatim:
          case Spadsrctxt:
            maxXvalue = max(maxXvalue, node->x + verbatimWidth(node));
            break;
          case Punctuation:
            maxXvalue = max(maxXvalue, node->x + punctuationWidth(node));
            break;
          case Dash:
            maxXvalue = max(maxXvalue, node->x + widthOfDash(node));
            break;
          case HSpace:
            maxXvalue = max(maxXvalue, node->x +
             (node->data.node != NULL ? atoi(node->data.node->data.text) : 1));
            break;
          case Space:
            maxXvalue =
             max(maxXvalue, node->x +
             (gTopOfGroupStack->cur_font->max_bounds.width) *
             (node->data.node != NULL ? atoi(node->data.node->data.text) : 1));
            break;
          case Group:
            pushGroupStack();
            break;
          case BoldFace:
            bfTopGroup();
            break;
```

```
            case Emphasize:
              if (gTopOfGroupStack->cur_font == gRmFont)
                  emTopGroup();
              else
                  rmTopGroup();
              break;
            case It:
              emTopGroup();
              break;
            case Rm:
            case Sl:
            case Tt:
              rmTopGroup();
              break;
            case Endgroup:
              popGroupStack();
              break;
            case Controlbitmap:
            case Inputbitmap:
              if (node->width == -1)
                  insertBitmapFile(node);
              maxXvalue = max(maxXvalue, node->x + node->width);
              break;
            case Inputpixmap:
              if (node->width == -1)
                  insertPixmapFile(node);
              maxXvalue = max(maxXvalue, node->y + node->width);
              break;
            default:
              break;
          }
      }
      return cur_height;
  }
```

9.13.51 Xvalue

⟨*hypertex*⟩+≡

```
static int Xvalue(TextNode * node) {
    for (; node != NULL; node = node->next) {
        switch (node->type) {
          case Controlbitmap:
          case Inputbitmap:
          case Inputpixmap:
          case Lsquarebrace:
          case Rsquarebrace:
          case Word:
          case Verbatim:
          case Spadsrctxt:
          case Dash:
          case Punctuation:
          case VSpace:
          case HSpace:
          case Horizontalline:
          case Box:
          case Downlink:
          case Link:
          case Lispwindowlink:
          case Lisplink:
          case Unixlink:
          case Spadcall:
          case Spadcallquit:
          case Qspadcall:
          case Qspadcallquit:
          case LispDownLink:
          case LispMemoLink:
          case Lispcommand:
          case Lispcommandquit:
          case Spadlink:
          case Spaddownlink:
          case Spadmemolink:
          case Spadcommand:
          case Spadgraph:
          case Unixcommand:
          case Space:
          case SimpleBox:
          case Radiobox:
            return node->x;
          default:
#ifdef DEBUG
            fprintf(stderr, "Xvalue did not know x value of type %d\n", node->typ
```

```
#endif
            return Xvalue(node->next);
        }
    }
    return 0;
}
```

9.13.52 trailingSpace

The trailingSpace function computes the length of the trailing spaces of a node.

⟨*hypertex*⟩+≡

```
int trailingSpace(TextNode * node) {
    int space = 0;
    for (; node->type < Endtokens; node = node->next);
    if (node->type == Space)
        space += inter_word_space *
            (node->data.node != NULL ? atoi(node->data.node->data.text) : 1);
    return space;
}
```

9.13.53 insertBitmapFile

The insertBitmapFile function reads a bitmap file into memory.

⟨*hypertex*⟩+≡

```
void insertBitmapFile(TextNode * node) {
    char *filename = node->data.text;
    int bm_width, bm_height;
    XImage *im;
    ImageStruct *image;
    if (*filename == ' ')
        filename++;
    if (node->image.pm == 0) {
        if (
        ((image = (ImageStruct *) hashFind(&gImageHashTable, filename))
            == NULL)  || (getenv("HTCACHE"))) {
            /*
             * read the bitmap if not already in memory or if the environment
             * variable HTCACHE is set (NAG addition).
             */
            im = HTReadBitmapFile(gXDisplay, gXScreenNumber, filename,
                                  &bm_width, &bm_height);
            /** now add the image to the gImageHashTable **/
            image = (ImageStruct *) halloc(sizeof(ImageStruct), "ImageStruct");
            image->image.xi = im;
            image->width = image->image.xi->width;
            image->height = image->image.xi->height;
            image->filename =
             (char *)halloc(sizeof(char) *strlen(filename)+1,"Image Filename");
            /* strcpy(image->filename, filename); */
            sprintf(image->filename, "%s", filename);
            hashInsert(&gImageHashTable, (char *)image, image->filename);
        }
        node->width = image->width;
        node->height = image->height;
        node->image.xi = image->image.xi;
    }
}
```

9.13.54 insertPixmapFile

The insertPixmapFile function reads a pixmap file into memory.

⟨*hypertex*⟩+≡

```
void insertPixmapFile(TextNode * node) {
    char *filename = node->data.text;
    int bm_width, bm_height, ret_val;
    XImage *xi;
    ImageStruct *image;

    if (*filename == ' ')
        filename++;
    if (node->image.xi == 0) {
      if ((image=(ImageStruct *)hashFind(&gImageHashTable, filename))==NULL) {
           ret_val = read_pixmap_file(gXDisplay, gXScreenNumber, filename, &xi,
                                      &bm_width, &bm_height);
            switch (ret_val) {
              case(-1):
                gSwitch_to_mono = 1;
                return;
              case BitmapFileInvalid:
                fprintf(stderr, "File %s contains invalid bitmap data\n",
                        filename);
                return;
              case BitmapOpenFailed:
                fprintf(stderr, "couldn't open bitmap file %s\n", filename);
                return;
              case BitmapNoMemory:
                fprintf(stderr, "not enough memory to store bitmap\n");
                return;
            }
            image = (ImageStruct *) halloc(sizeof(ImageStruct), "ImageStruct");
            image->width = bm_width;
            image->height = bm_height;
            image->filename = (char *)halloc(sizeof(char) *strlen(filename)+1,
                                             "insert_pixmap--filename");
            /* strcpy(image->filename, filename); */
            sprintf(image->filename, "%s", filename);
            image->image.xi = xi;
            hashInsert(&gImageHashTable, (char *)image, image->filename);
        }
        node->width = image->width;
        node->height = plh(image->height + inter_line_space);
        node->image.xi = image->image.xi;
    }
```

```
}
```

9.13.55 plh

The plh function calculates the closet value of line_height ¿ height.

⟨*hypertex*⟩+≡

```
int plh(int height) {
    int rheight = height;
    if (gExtentRegion == Scrolling) {
        for (rheight = line_height; rheight < height; rheight += line_height)
          ;
    }
    return rheight;
}
```

9.14 Handling forms

A few routines used to help with form extents

9.14.1 computeFormPage

To solve the problem of improperly nested `\em`, I will have to keep and always initialize the top of the stack.

⟨*hypertex*⟩+≡

```
void computeFormPage(HyperDocPage *page) {
    while (popGroupStack() >= 0);
    /*
     * The compute the text extents
     */
    formHeaderExtent(page);
    formFooterExtent(page);
    formScrollingExtent(page);
    gWindow->height = windowHeight(gWindow->page);
}
```

9.14.2 windowWidth

A simple function that returns the width needed to store show the number of columns given.

⟨*hypertex*⟩+≡
```
int windowWidth(int cols) {
    return (left_margin + cols * space_width + non_scroll_right_margin_space);
}
```

9.14.3 windowHeight

⟨*hypertex*⟩+≡
```
static int windowHeight(HyperDocPage *page) {
    int temp;
    temp = page->header->height + top_margin + bottom_margin;
    if (page->scrolling)
        temp += page->scrolling->height + page->footer->height;
    return (temp);
}
```

9.14.4 formHeaderExtent

⟨*hypertex*⟩+≡
```
static void formHeaderExtent(HyperDocPage *page) {
    /*
     * Hopefully I will soon be able to actually compute the needed height
     * for the header here
     */
    gExtentRegion = Header;
    right_margin_space = non_scroll_right_margin_space;
    initExtents();
    text_y = top_margin + line_height;
    computeTextExtent(page->header->next);
    page->header->height = (gInLine) ? text_y : text_y - past_line_height;
    if (!(page->pageFlags & NOLINES))
        page->header->height += (int) line_height / 2;
    page->header->height += gWindow->border_width;
}
```

9.14.5 formFooterExtent

⟨*hypertex*⟩+≡

```
static void formFooterExtent(HyperDocPage *page) {
    if (page->footer) {
        gExtentRegion = Footer;
        right_margin_space = non_scroll_right_margin_space;
        initExtents();
        computeTextExtent(page->footer->next);
        /*
         * I inserted the 2nd arg to textHeight below because it
         * was missing. Perhaps there is a better value for it.
         */
        page->footer->height = textHeight(page->footer->next,
            page->footer->next->type);
        if ((!page->pageFlags & NOLINES))
            page->footer->height += (int) line_height / 2;
    }
}
```

9.14.6 formScrollingExtent

⟨*hypertex*⟩+≡

```
static void formScrollingExtent(HyperDocPage *page) {
    /*
     * Check to see if there is a scrolling region
     */
    if (page->scrolling) {
        /*
         * If there is then compute all the proper locations
         */
        gExtentRegion = Scrolling;
        right_margin_space = non_scroll_right_margin_space + gScrollbarWidth;
        initExtents();
        text_y = line_height;
        computeTextExtent(page->scrolling->next);
        if (!gInLine)
            text_y = text_y - past_line_height;
        else if (present_line_height > line_height)
            text_y = text_y + present_line_height - line_height;
        page->scrolling->height = text_y;
    }
}
```

9.15 Managing the HyperDoc group stack

9.15.1 popGroupStack

This routine pops the top of the current group stack.

⟨*hypertex*⟩+≡

```
int popGroupStack(void) {
    GroupItem *junk;
    /*
     * If the the stack has only a single item, then pop it anyway so the
     * user can see the problem
     */
    if (! gTopOfGroupStack->next)
        return -1;
    /* Else, Pop the thing */
    junk = gTopOfGroupStack;
    gTopOfGroupStack = gTopOfGroupStack->next;
    junk->next = NULL;
    free(junk);
    /* Now change the font to the cur_font and the cur_color */
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
    return 1;
}
```

9.15.2 pushGroupStack

⟨*hypertex*⟩+≡

```
void pushGroupStack(void) {
    /*
     * This routine makes room by pushing a new item on the stack
     */
    GroupItem *newgp;
    newgp = (GroupItem *) halloc(sizeof(GroupItem), "Push Group Stack");
    newgp->cur_font = gTopOfGroupStack->cur_font;
    newgp->cur_color = gTopOfGroupStack->cur_color;
    newgp->center = gTopOfGroupStack->center;
    newgp->next = gTopOfGroupStack;
    gTopOfGroupStack = newgp;
}
```

9.15.3 initGroupStack

⟨*hypertex*⟩+≡

```
void initGroupStack(void) {
    gTopOfGroupStack =
      (GroupItem *) halloc(sizeof(GroupItem), "Push Group Stack");
    gTopOfGroupStack->center = 0;
    gTopOfGroupStack->next = NULL;
    gTopOfGroupStack->cur_color = 0;
    gTopOfGroupStack->cur_font = NULL;
}
```

9.15.4 emTopGroup

⟨*hypertex*⟩+≡

```
void emTopGroup(void) {
    if (! gTopOfGroupStack->next)
        pushGroupStack();
    gTopOfGroupStack->cur_color = gEmColor;
    gTopOfGroupStack->cur_font = gEmFont;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

9.15.5 rmTopGroup

⟨*hypertex*⟩+≡

```
void rmTopGroup(void) {
    if (! gTopOfGroupStack->next)
        pushGroupStack();
    gTopOfGroupStack->cur_color = gRmColor;
    gTopOfGroupStack->cur_font = gRmFont;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

9.15.6 lineTopGroup

⟨*hypertex*⟩+≡
```
void lineTopGroup(void) {
    if (! gTopOfGroupStack->next)
        pushGroupStack();
    gTopOfGroupStack->cur_color = gBorderColor;
    gTopOfGroupStack->cur_font = gRmFont;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);

}
```

9.15.7 bfTopGroup

⟨*hypertex*⟩+≡
```
void bfTopGroup(void) {
    /*
     * Just in case the person is tryin a \em without a grouping
     */
    if (! gTopOfGroupStack->next)
        pushGroupStack();
    gTopOfGroupStack->cur_color = gBfColor;
    gTopOfGroupStack->cur_font = gBfFont;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

9.15.8 ttTopGroup

⟨*hypertex*⟩+≡
```
void ttTopGroup(void) {
    if (! gTopOfGroupStack->next)
        pushGroupStack();
    gTopOfGroupStack->cur_color = gTtColor;
    gTopOfGroupStack->cur_font = gTtFont;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

9.15.9 pushActiveGroup

⟨*hypertex*⟩+≡
```
void pushActiveGroup(void) {
    pushGroupStack();
    gTopOfGroupStack->cur_font = gActiveFont;
    gTopOfGroupStack->cur_color = gActiveColor;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

9.15.10 pushSpadGroup

⟨*hypertex*⟩+≡
```
void pushSpadGroup(void) {
    pushGroupStack();
    gTopOfGroupStack->cur_font = gAxiomFont;
    gTopOfGroupStack->cur_color = gAxiomColor;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

9.15.11 initTopGroup

⟨*hypertex*⟩+≡
```
void initTopGroup(void) {
    /* clear the group stack */
    while (popGroupStack() >= 0)
        ;
    /* then set the colors to be normal */
    gTopOfGroupStack->cur_color = gRmColor;
    gTopOfGroupStack->cur_font = gRmFont;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

9.15.12 centerTopGroup

⟨*hypertex*⟩+≡
```
void centerTopGroup(void) {
    pushGroupStack();
    gTopOfGroupStack->center = 1;
}
```

9.15.13 copyGroupStack

⟨*hypertex*⟩+≡

```
GroupItem *copyGroupStack(void) {
    GroupItem *newgp = NULL;
    GroupItem *first = NULL;
    GroupItem *prev  = NULL;
    GroupItem *trace = gTopOfGroupStack;
    while (trace) {
        newgp = (GroupItem *) halloc(sizeof(GroupItem), "Copy Group Stack");
        newgp->cur_font = trace->cur_font;
        newgp->cur_color = trace->cur_color;
        newgp->center = trace->center;
        if (!first)
            first = newgp;
        else
            prev->next = newgp;
        prev = newgp;
        trace = trace->next;
    }
    if (newgp)
        newgp->next = NULL;
    return first;
}
```

9.15.14 freeGroupStack

⟨*hypertex*⟩+≡

```
void freeGroupStack(GroupItem *g) {
    GroupItem *trace = g;
    while (trace) {
        GroupItem *junk = trace;
        trace = trace->next;
        free(junk);
    }
}
```

9.16 Handle input, output, and Axiom communication

9.16.1 makeRecord

⟨*hypertex*⟩+≡

```
void makeRecord(void) {
  int i;
  for (i=0;i<input_file_count;i++){
    sendLispCommand("(|clearCmdCompletely|)");
    sendLispCommand("(setq |$testingSystem| T)");
    sendLispCommand("(setq |$printLoadMsgs| NIL)");
    sendLispCommand("(setq |$BreakMode| '|resume|)");
    sprintf(buf_for_record_commands,
       "(|inputFile2RecordFile| '\"%s\")",input_file_list[i]);
    fprintf(stderr,"%s\n",buf_for_record_commands);
    sendLispCommand(buf_for_record_commands);
  }
  if (kill_spad){
    i = connectSpad();
    if (i != NotConnected && i != SpadBusy)
      send_int(spadSocket, KillLispSystem);
  }
}
```

9.16.2 verifyRecord

⟨*hypertex*⟩+≡

```
void verifyRecord(void) {
  int i;
  for (i=0;i<input_file_count;i++){
    sendLispCommand("(|clearCmdCompletely|)");
    sendLispCommand("(setq |$testingSystem| T)");
    sendLispCommand("(setq |$printLoadMsgs| NIL)");
    sendLispCommand("(setq |$BreakMode| '|resume|)");
    sprintf(buf_for_record_commands,
      "(|verifyRecordFile| '\"%s\")",input_file_list[i]);
    fprintf(stderr,"%s\n",buf_for_record_commands);
    sendLispCommand(buf_for_record_commands);
  }
  if (kill_spad) {
    i = connectSpad();
    if (i != NotConnected && i != SpadBusy)
      send_int(spadSocket, KillLispSystem);
  }
}
```

9.16.3 ht2Input

⟨*hypertex*⟩+≡

```
void ht2Input(void) {
  HashTable *table;
  HashEntry *entry;
  int i;
  bsdSignal(SIGUSR2, SIG_IGN,RestartSystemCalls);
  gWindow = allocHdWindow();
  initGroupStack();
  table = gWindow->fPageHashTable;
  makeInputFileList();
  for (i = 0; i < table->size; i++)
    for (entry = table->table[i]; entry != NULL; entry = entry->next)
      makeTheInputFile((UnloadedPage *) entry->data);
  if (kill_spad){
    i = connectSpad();
    if (i != NotConnected && i != SpadBusy)
      send_int(spadSocket, KillLispSystem);
  }
}
```

9.16.4 makeInputFileName

⟨*hypertex*⟩+≡

```
static char *makeInputFileName(char *buf, char *filename) {
    char *b, *c;
    strcpy(buf, filename);
    for (b = buf + strlen(buf) - 1; b != buf && *b != '/'; b--);
    if (b != buf)
        b = b + 1;
    for (c = b; *c != '.' || c[1] != 'h' || c[2] != 't'; c++);
    strcpy(c, ".input");
    return b;
}
```

9.16.5 makePasteFileName

⟨*hypertex*⟩+≡

```
static char *makePasteFileName(char *buf, char *filename) {
    char *b, *c;
    strcpy(buf, filename);
    for (b = buf + strlen(buf) - 1; b != buf && *b != '/'; b--);
    if (b != buf)
        b = b + 1;
    for (c = b; *c != '.' || c[1] != 'h' || c[2] != 't'; c++);
    strcpy(c, ".pht");
    return b;
}
```

9.16.6 makeTheInputFile

⟨*hypertex*⟩+≡

```
static void makeTheInputFile(UnloadedPage *page) {
    char buf[1024], *b;
    if (!page->fpos.name)
        return;
    b = makeInputFileName(buf, page->fpos.name);
    if (inListAndNewer(b, page->fpos.name)) {
        printf("parsing: %s\n", page->name);
        if (setjmp(jmpbuf)) {
            printf("Syntax error!\n");
        }
        else {
            loadPage((HyperDocPage *)page);
            makeInputFileFromPage(gWindow->page);
        }
    }
}
```

9.16.7 makeInputFileFromPage

⟨*hypertex*⟩+≡

```
static void makeInputFileFromPage(HyperDocPage *page) {
  TextNode *node;
  int starting_file = 1,/* i,*/ /*len,*/ ret_val;
  char *buf, buf2[1024], buf3[1024];
  char *b, *c, *com;
  FILE *file = NULL;
  FILE *pfile = NULL;
  static HyperDocPage *op = NULL;
  if (op == page)
    return;
  op = page;
  if (page == NULL)
    return;
  b = makeInputFileName(buf2, page->filename);
  c = makePasteFileName(buf3, page->filename);
  if (inListAndNewer(b, page->filename)) {
    /* open and prepare the input file */
    file = fopen(b, "a");
    if (file == NULL) {
      fprintf(stderr, "couldn't open output file %s\n", b);
      exit(-1);
    }
    fprintf(file, "\n-- Input for page %s\n", page->name);
    fprintf(file, ")clear all\n\n");
    for (node = page->scrolling; node != NULL; node = node->next)
      if (node->type == Spadcommand || node->type == Spadgraph
          || node->type == Spadsrc) {
        if (starting_file) {
          example_number = 1;
          if (make_patch_files) {
            sendLispCommand("(|clearCmdAll|)");
            sendLispCommand("(|resetWorkspaceVariables|)");
            sendLispCommand("(setq $linelength 55)");
            sendLispCommand("(|setOutputCharacters| '(default))");
            sendLispCommand("(setq |$printLoadMsgs| NIL)");
            sendLispCommand("(setq |$UserLevel| '|development|)");
            sendLispCommand("(verbos 0)");
          }
          if (make_patch_files) {
            pfile = fopen(c, "a");
            if (pfile == NULL) {
              fprintf(stderr, "couldn't open output file %s\n", c);
              exit(-1);
```

```
            }
          }
          starting_file = 0;
        }
        else
          example_number++;
        buf = printToString(node->next);
        com = allocString(buf);
        fprintf(file, "%s\n", buf);
        fflush(file);
        fprintf(stderr, "writing:\t%s\n", buf);
        include_bf = 1;
        buf = printToString(node->next);
        include_bf = 0;
        if (make_patch_files) {
          if (node->type == Spadcommand || node->type == Spadsrc)
            printPaste(pfile, com, buf, page->name, node->type);
          else
            printGraphPaste(pfile, com, buf, page->name, node->type);
        }
      }
    if (!starting_file && make_patch_files) {
      ret_val = fclose(pfile);
      if (ret_val == -1) {
        fprintf(stderr, "couldn't close file %s\n", b);
        exit(-1);
      }
    }
    ret_val = fclose(file);
    if (ret_val == -1) {
      fprintf(stderr, "couldn't close file %s\n", b);
      exit(-1);
    }
  }
}
```

9.16.8 strCopy

⟨*hypertex*⟩+≡

```
char *strCopy(char *s) {
    char *b = halloc(strlen(s) + 1,"String");
    strcpy(b, s);
    return b;
}
```

9.16.9 inListAndNewer

⟨*hypertex*⟩+≡

```
static int inListAndNewer(char *inputFile, char *htFile) {
    int ret_val, found = 0, i;
    struct stat htBuf, inputBuf;
    for (i = 0; i < num_active_files; i++) {
        if (strcmp(active_file_list[i], inputFile) == 0) {
            found = 1;
            break;
        }
    }
    if (found)
        return 1;
    found = 0;
    for (i = 0; i < num_inactive_files; i++)
        if (strcmp(inactive_file_list[i], inputFile) == 0) {
            found = 1;
            break;
        }
    if (found)
        return 0;
    found = 0;
    for (i = 0; i < input_file_count; i++)
        if (strcmp(input_file_list[i], inputFile) == 0) {
            found = 1;
            break;
        }
    if (!found) {
        inactive_file_list[num_inactive_files++] = strCopy(inputFile);
        return 0;
    }
    ret_val = stat(inputFile, &inputBuf);
    if (ret_val == -1) {
        active_file_list[num_active_files++] = input_file_list[i];
        printf("making %s\n", inputFile);
        return 1;
    }
    ret_val = stat(htFile, &htBuf);
    if (ret_val == -1) {
        inactive_file_list[num_inactive_files++] = strCopy(inputFile);
        return 0;
    }
    ret_val = htBuf.st_mtime > inputBuf.st_mtime;
    ret_val = 1;
    if (ret_val) {
```

```
        active_file_list[num_active_files++] = input_file_list[i];
        printf("making %s\n", inputFile);
        unlink(inputFile);
    }
    else
        inactive_file_list[num_inactive_files++] = input_file_list[i];
    return ret_val;
}
```

9.16.10 makeInputFileList

⟨*hypertex*⟩+≡

```
static void makeInputFileList(void) {
    int i;
    char buf[256], *name;
    for (i = 0; i < input_file_count; i++) {
        name = makeInputFileName(buf, input_file_list[i]);
        input_file_list[i] = (char *)halloc(strlen(name) + 1,"Input Filename");
        strcpy(input_file_list[i], name);
    }
}
```

9.16.11 printPasteLine

⟨*hypertex*⟩+≡

```
void printPasteLine(FILE *pfile,char *str) {
    char *free = "\\free", *bound = "\\bound", *f = free, *b = bound;
    int justSaw = 0;
    for (; *str; str++) {
        if (*f == '\0')
            justSaw = 2;
        if (*b == '\0')
            justSaw = 2;
        if (*b == *str)
            b++;
        else
            b = bound;
        if (*f == *str)
            f++;
        else
            f = free;
        if (*str == '%' || *str == '{' || *str == '}' || *str == '#') {
            if (*str == '{' && justSaw)
                justSaw--;
            else if (*str == '}' && justSaw)
                justSaw--;
            else
                putc('\\', pfile);
        }
        putc(*str, pfile);
    }
}
```

9.16.12 getSpadOutput

⟨*hypertex*⟩+≡

```
void getSpadOutput(FILE *pfile,char *command,int com_type) {
    int n, i;
    char buf[1024];
    sendCommand(command, com_type);
    n = get_int(spadSocket);
    for (i = 0; i < n; i++) {
        get_string_buf(spadSocket, buf, 1024);
        fprintf(pfile, "%s\n", buf);
    }
    unescapeString(command);
}
```

9.16.13 getGraphOutput

THEMOS says: There is a problem here in that we issue the (—close—) and then go on. If this is the last command ,we will soon send a SIGTERM and the whole thing will collapse maybe BEFORE the writing out has finished. Fix: Call a Lisp function that checks (with `\axiomOp{key}` ps and grep) the health of the viewport. We do this after the (—close—).

⟨*hypertex*⟩+≡

```
void getGraphOutput(char *command,char *pagename,int com_type) {
    int n, i;
    char buf[1024];
    sendCommand(command, com_type);
    n = get_int(spadSocket);
    for (i = 0; i < n; i++) {
        get_string_buf(spadSocket, buf, 1024);
    }
    unescapeString(command);
    sprintf(buf,
         "(|processInteractive| '(|write| |%s| \"%s%d\" \"image\") NIL)", "%",
          pagename, example_number);
    sendLispCommand(buf);
    sendLispCommand("(|setViewportProcess|)");
    sendLispCommand("(|processInteractive| '(|close| (|%%| -3)) NIL)");
    sendLispCommand("(|waitForViewport|)");
    get_int(spadSocket);
}
```

9.16.14 sendCommand

⟨*hypertex*⟩+≡

```
static void sendCommand(char *command,int com_type) {
    char buf[1024];
    if (com_type != Spadsrc) {
        escapeString(command);
        sprintf(buf, "(|parseAndEvalToHypertex| '\"%s\")", command);
        sendLispCommand(buf);
    }
    else {
        FILE *f;
        char name[512], str[512]/*, *c*/;
        sprintf(name, "/tmp/hyper%s.input", getenv("SPADNUM"));
        f = fopen(name, "w");
        if (f == NULL) {
            fprintf(stderr, "Can't open temporary input file %s\n", name);
            return;
        }
        fprintf(f, "%s", command);
        fclose(f);
        sprintf(str, "(|parseAndEvalToHypertex| '\")read %s\")", name);
        sendLispCommand(str);
    }
}
```

9.16.15 printPaste

⟨*hypertex*⟩+≡

```
static void printPaste(FILE *pfile,char *realcom,char *command,
            char *pagename,int com_type) {
    fprintf(pfile, "\\begin{patch}{%sPatch%d}\n", pagename, example_number);
    fprintf(pfile, "\\begin{paste}{%sFull%d}{%sEmpty%d}\n",
            pagename, example_number, pagename, example_number);
    fprintf(pfile, "\\pastebutton{%sFull%d}{\\hidepaste}\n",
            pagename, example_number);
    fprintf(pfile, "\\tab{5}\\spadcommand{");
    printPasteLine(pfile, command);
    fprintf(pfile, "}\n");
    fprintf(pfile, "\\indentrel{3}\\begin{verbatim}\n");
    getSpadOutput(pfile, realcom, com_type);
    fprintf(pfile, "\\end{verbatim}\n");
    fprintf(pfile, "\\indentrel{-3}\\end{paste}\\end{patch}\n\n");

    fprintf(pfile, "\\begin{patch}{%sEmpty%d}\n", pagename, example_number);
    fprintf(pfile, "\\begin{paste}{%sEmpty%d}{%sPatch%d}\n",
            pagename, example_number, pagename, example_number);
    fprintf(pfile, "\\pastebutton{%sEmpty%d}{\\showpaste}\n",
            pagename, example_number);
    fprintf(pfile, "\\tab{5}\\spadcommand{");
    printPasteLine(pfile, command);
    fprintf(pfile, "}\n");
    fprintf(pfile, "\\end{paste}\\end{patch}\n\n");
    fflush(pfile);
}
```

9.16.16 printGraphPaste

⟨*hypertex*⟩+≡

```
static void printGraphPaste(FILE *pfile,char *realcom,
                  char *command,char *pagename,int com_type) {
    fprintf(pfile, "\\begin{patch}{%sPatch%d}\n", pagename, example_number);
    fprintf(pfile, "\\begin{paste}{%sFull%d}{%sEmpty%d}\n",
            pagename, example_number, pagename, example_number);
    fprintf(pfile, "\\pastebutton{%sFull%d}{\\hidepaste}\n",
            pagename, example_number);
    fprintf(pfile, "\\tab{5}\\spadgraph{");
    printPasteLine(pfile, command);
    fprintf(pfile, "}\n");
    fprintf(pfile, "\\center{\\unixcommand{\\inputimage{\\env{AXIOM}");
    fprintf(pfile, "/doc/viewports/%s%d.view/image}}",
                pagename,example_number);
    fprintf(pfile, "{viewalone\\space{1} \\env{AXIOM}");
    fprintf(pfile,"/doc/viewports/%s%d}}\n", pagename, example_number);
    getGraphOutput(realcom, pagename, com_type);
    fprintf(pfile, "\\end{paste}\\end{patch}\n\n");

    fprintf(pfile, "\\begin{patch}{%sEmpty%d}\n", pagename, example_number);
    fprintf(pfile, "\\begin{paste}{%sEmpty%d}{%sPatch%d}\n",
            pagename, example_number, pagename, example_number);
    fprintf(pfile, "\\pastebutton{%sEmpty%d}{\\showpaste}\n",
            pagename, example_number);
    fprintf(pfile, "\\tab{5}\\spadgraph{");
    printPasteLine(pfile, command);
    fprintf(pfile, "}\n");
    fprintf(pfile, "\\end{paste}\\end{patch}\n\n");
    fflush(pfile);
}
```

9.17 X Window window initialization code

Initialize the X Window System.

9.17.1 initializeWindowSystem

⟨*hypertex*⟩+≡

```
void initializeWindowSystem(void) {
    char *display_name = NULL;
    XColor fg, bg;
    Colormap cmap;
    Pixmap  mousebits, mousemask;
/*    fprintf(stderr,"initx:initializeWindowSystem:entered\n");*/
    /* Try to open the display */
/*    fprintf(stderr,"initx:initializeWindowSystem:XOpenDisplay\n");*/
    if ((gXDisplay = XOpenDisplay(display_name)) == NULL) {
        fprintf(stderr, "(HyperDoc) Cannot connect to the X11 server!\n");
        exit(-1);
    }
    /* Get the screen */
/*    fprintf(stderr,"initx:initializeWindowSystem:DefaultScreen\n");*/
    gXScreenNumber = scrn = DefaultScreen(gXDisplay);
/*    fprintf(stderr,"initx:initializeWindowSystem:XGContextFromGC\n");*/
    server_font =XGContextFromGC(DefaultGC(gXDisplay, gXScreenNumber));
    /* Get the cursors we need. */
/*    fprintf(stderr,"initx:initializeWindowSystem:DefaultColormap\n");*/
    cmap = DefaultColormap(gXDisplay, gXScreenNumber);
/*    fprintf(stderr,"initx:initializeWindowSystem:WhitePixel\n");*/
    fg.pixel = WhitePixel(gXDisplay,gXScreenNumber);
/*    fprintf(stderr,"initx:initializeWindowSystem:XQueryColor\n");*/
    XQueryColor(gXDisplay, cmap, &fg );
/*    fprintf(stderr,"initx:initializeWindowSystem:BlackPixel\n");*/
    bg.pixel = BlackPixel(gXDisplay,gXScreenNumber);
/*    fprintf(stderr,"initx:initializeWindowSystem:XQueryColor2\n");*/
    XQueryColor(gXDisplay, cmap, &bg );
/*  fprintf(stderr,"initx:initializeWindowSystem:XCreateBitmapFromData 1\n");*/
    mousebits = XCreateBitmapFromData(gXDisplay,
        RootWindow(gXDisplay, gXScreenNumber),
        mouseBitmap_bits, mouseBitmap_width,mouseBitmap_height);
/* fprintf(stderr,"initx:initializeWindowSystem:XCreateBitmapFromData 2\n");*/
    mousemask = XCreateBitmapFromData(gXDisplay,
        RootWindow(gXDisplay, gXScreenNumber),
        mouseMask_bits, mouseMask_width,mouseMask_height);
/* fprintf(stderr,"initx:initializeWindowSystem:XCreateBitmapFromData 2\n");*/
    gActiveCursor = XCreatePixmapCursor(gXDisplay,
```

```
        mousebits, mousemask, &fg, &bg,
        mouseBitmap_x_hot,mouseBitmap_y_hot);
/*    fprintf(stderr,"initx:initializeWindowSystem:XCreateFontCursor\n");*/
    gNormalCursor = XCreateFontCursor(gXDisplay, XC_left_ptr);
/*    fprintf(stderr,"initx:initializeWindowSystem:XCreateFontCursor 2\n");*/
    gBusyCursor = XCreateFontCursor(gXDisplay, XC_watch);
    /* Now initialize all the colors and fonts */
/*    fprintf(stderr,"initx:initializeWindowSystem:ingItColorsAndFonts\n");*/
    ingItColorsAndFonts();
/*    fprintf(stderr,"initx:initializeWindowSystem:initText\n");*/
    initText();
/*    fprintf(stderr,"initx:initializeWindowSystem:exited\n");*/
}
```

9.17.2 initTopWindow

This routine is responsible for initializing a HyperDoc Window. At this point, all the fonts have been loaded, and X has been initialized. All I need worry about is starting up the window, and creating some of its children.

The initTopWindow function tries to start up a window with the page name. If the page name is NULL, it doesn't try to find it in the Hash Table, but rather just allocates a page of no name

⟨*hypertex*⟩+≡

```
int initTopWindow(char *name) {
    HyperDocPage *page;
    XSetWindowAttributes wa;    /* The X attributes structure */
    HDWindow *old_win = gWindow;
    gWindow = allocHdWindow();
    if (name == NULL) {
        /** Then allocate an empty page, and assign it to gWindow->page */
        page = allocPage((char *) NULL);
    }
    else {
        /* Try to find the page in the page hash table */
        page = (HyperDocPage *) hashFind(gWindow->fPageHashTable, name);
        if (page == NULL) {
            fprintf(stderr,
                  "(HyperDoc) Couldn\'t find page %s in page hash table \n",
                   name);
            if (gParentWindow == NULL)
                /* Gaak, This is a start up error */
                exit(-1);
            else {
                gWindow = old_win;
                return -1;
            }
        }
    }
    /* First allocate memory for the new window structure   */
    gWindow->page = page;
    if (old_win == NULL)
        openWindow(0);
    else
        openWindow(old_win->fMainWindow);
    getGCs(gWindow);
    XMapWindow(gXDisplay, gWindow->fMainWindow);
    hashInsert(&gSessionHashTable, (char *)gWindow,
               (char *) &gWindow->fMainWindow);
    changeText(gRmColor, gRmFont);
```

```
    wa.background_pixel = gBackgroundColor;
    XChangeWindowAttributes(gXDisplay, gWindow->fMainWindow, CWBackPixel, &wa);
    XChangeWindowAttributes(gXDisplay,gWindow->fScrollWindow,CWBackPixel,&wa);
    return 1;
}
```

9.17.3 openFormWindow

Create and initialize a form HyperDoc window.

⟨*hypertex*⟩+≡

```
static void openFormWindow(void) {
    int x, y, width, height;
    unsigned int fwidth = 0, fheight = 0;
    unsigned int xadder = 0, yadder = 0;
    /*char *window_name = "HyperDoc";*/
    /*char *icon_name = "HT";*/
    XrmValue value;
    char *str_type[50];
    XSizeHints size_hints;
    int userSpecified = 0;
    char userdefaults[50], progdefaults[50];
    strcpy(progdefaults, "=950x450+0+0");
    if (XrmGetResource(rDB, "Axiom.hyperdoc.FormGeometry",
        "Axiom.hyperdoc.FormGeometry", str_type, &value) == True)
    {
        strncpy(userdefaults, value.addr, (int) value.size);
        userSpecified = 1;
    }
    else
        strcpy(userdefaults, progdefaults);
    XGeometry(gXDisplay, gXScreenNumber, userdefaults, progdefaults,
              0, fwidth, fheight, xadder, yadder,
              &x, &y, &width, &height);
    gWindow->border_width = getBorderProperties();
    gWindow->width = 1;
    gWindow->height = 1;
    gWindow->fMainWindow =
     XCreateSimpleWindow(gXDisplay, RootWindow(gXDisplay, gXScreenNumber),
                         x, y, width, height, gWindow->border_width,
                         gBorderColor, WhitePixel(gXDisplay, gXScreenNumber));
    gWindow->fScrollWindow =
      XCreateSimpleWindow(gXDisplay, gWindow->fMainWindow, 1, 1, 1, 1, 0,
                          BlackPixel(gXDisplay, gXScreenNumber),
                          WhitePixel(gXDisplay, gXScreenNumber));
    makeScrollBarWindows();
    makeTitleBarWindows();
    setNameAndIcon();
    XSelectInput(gXDisplay, gWindow->fScrollWindow, PointerMotionMask);
    XSelectInput(gXDisplay, gWindow->fMainWindow,
                 StructureNotifyMask | PointerMotionMask);
    XDefineCursor(gXDisplay, gWindow->fMainWindow, gNormalCursor);
```

```
    /* now give the window manager some hints */
    size_hints.flags = 0;
    size_hints.min_width  = width;
    size_hints.min_height = height;
    size_hints.flags |= PMinSize;
    size_hints.width  = width;
    size_hints.height = height;
    size_hints.flags |= (userSpecified ? USSize : PSize);
    size_hints.x = x;
    size_hints.y = y;
    size_hints.flags |= (userSpecified ? USPosition : PPosition);
    XSetNormalHints(gXDisplay, gWindow->fMainWindow, &size_hints);
    XFlush(gXDisplay);
}
```

9.17.4 initFormWindow

⟨*hypertex*⟩+≡

```
int initFormWindow(char *name, int cols) {
    XSetWindowAttributes wa;    /* The X attributes structure */
    /* First allocate memory for the new window structure   */
    gWindow = allocHdWindow();
    openFormWindow();
    gWindow->width = windowWidth(cols);
    if (name == NULL) {
        /** Then allocate an empty page, and assign it to gWindow->page */
        gWindow->page = allocPage((char *) NULL);
    }
    else {
        /* Try to find the page in the page hash table */
        gWindow->page=(HyperDocPage *)hashFind(gWindow->fPageHashTable, name);
        if (gWindow->page == NULL) {
            fprintf(stderr, "Couldn't find page %s\n", name);
            return (-1);
        }
    }
    getGCs(gWindow);
    hashInsert(&gSessionHashTable, (char *)gWindow,
               (char *) &gWindow->fMainWindow);
    wa.background_pixel = gBackgroundColor;
    XChangeWindowAttributes(gXDisplay, gWindow->fMainWindow, CWBackPixel, &wa);
    XChangeWindowAttributes(gXDisplay,gWindow->fScrollWindow,CWBackPixel,&wa);
    return 1;
}
```

9.17.5 setNameAndIcon

⟨*hypertex*⟩+≡

```
static void setNameAndIcon(void) {
    char *icon_name = "HyperDoc";
    char *s;
    Pixmap icon_pixmap;
    XWMHints wmhints;
    XClassHint ch;
    ch.res_name = "HyperDoc";
    ch.res_class = gArgv[0];
    for (s = gArgv[0] + strlen(gArgv[0]) - 1; s != gArgv[0]; s--) {
        if (*s == '/') {
            ch.res_class = s + 1;
            break;
        }
    }
    XSetClassHint(gXDisplay, gWindow->fMainWindow, &ch);
    XStoreName(gXDisplay, gWindow->fMainWindow, "HyperDoc");
    /* define and assign the pixmap for the icon */
    icon_pixmap =
      XCreateBitmapFromData(gXDisplay, gWindow->fMainWindow, ht_icon_bits,
                            ht_icon_width, ht_icon_height);
    wmhints.icon_pixmap = icon_pixmap;
    wmhints.flags = IconPixmapHint;
    XSetWMHints(gXDisplay, gWindow->fMainWindow, &wmhints);
    /* name the icon */
    XSetIconName(gXDisplay, gWindow->fMainWindow, icon_name);
}
```

9.17.6 getBorderProperties

⟨*hypertex*⟩+≡

```
static int getBorderProperties(void) {
    char *bwidth;
    int bw;
    Colormap cmap;
    bwidth = "2";
    if (bwidth == NULL)
        bw = 1;
    else {
        bw = atoi(bwidth);
        if (bw < 1) {
            fprintf(stderr,
                "%s: The line width value must be greater than zero\n",
                "Axiom.hyperdoc");
            bw = 1;
        }
    }
    /* Now try to find the user preferred border color */
    if (DisplayPlanes(gXDisplay, gXScreenNumber) == 1)
        gBorderColor = BlackPixel(gXDisplay, gXScreenNumber);
    else {
        cmap = DefaultColormap(gXDisplay, gXScreenNumber);
        gBorderColor = getColor("BorderColor", "Foreground",
            BlackPixel(gXDisplay, gXScreenNumber), &cmap);
    }
    return bw;
}
```

9.17.7 openWindow

Create and initialize the HyperDoc window.

⟨*hypertex*⟩+≡

```
static void openWindow(Window w) {
    int x = 0, y = 0;
    /*int border_width = 2;*/
    unsigned int width = 1;
    unsigned int height = 1;
    unsigned int fwidth = 0, fheight = 0;
    unsigned int xadder = 0, yadder = 0;
    char *str_type[50];
    XrmValue value;
    char userdefaults[50], progdefaults[50];
    strcpy(progdefaults, "=700x450+0+0");
    if (XrmGetResource(rDB, "Axiom.hyperdoc.Geometry",
        "Axiom.hyperdoc.Geometry", str_type, &value) == True)
    {
        strncpy(userdefaults, value.addr, (int) value.size);
    }
    else
        strcpy(userdefaults, progdefaults);
    XGeometry(gXDisplay, gXScreenNumber, userdefaults, progdefaults,
              0, fwidth, fheight, xadder, yadder,
              &x, &y, ( int *)&width,( int *) &height);
    gWindow->border_width = getBorderProperties();
    gWindow->fMainWindow =
      XCreateSimpleWindow(gXDisplay, RootWindow(gXDisplay, gXScreenNumber),
                                  x, y, width, height, gWindow->border_width,
                                  gBorderColor,
                                  WhitePixel(gXDisplay, gXScreenNumber));
    gWindow->fScrollWindow =
     XCreateSimpleWindow(gXDisplay, gWindow->fMainWindow, 1, 1, 1, 1, 0,
                         gBorderColor, WhitePixel(gXDisplay, gXScreenNumber));
    makeScrollBarWindows();
    makeTitleBarWindows();
    /* Now set all the little properties for the top level window */
    setNameAndIcon();
    setSizeHints(w);
    XSelectInput(gXDisplay, gWindow->fScrollWindow, PointerMotionMask);
    XSelectInput(gXDisplay, gWindow->fMainWindow,
                 StructureNotifyMask | PointerMotionMask);
    XDefineCursor(gXDisplay, gWindow->fMainWindow, gNormalCursor);
}
```

9.17.8 setSizeHints

This routine gets and sets the size for a new window. If the w paramter is null, it means that this is the initial window. Thus the user preferences are checked. If this is not the first window, then the window w is used as a guidline, and the new window is placed on top of it.

⟨*hypertex*⟩+≡

```
static void setSizeHints(Window w) {
    int x, y;
    unsigned int width, height;
    char userdefaults[50];
    char progdefaults[50];
    char *str_type[50];
    unsigned int fwidth = 0, fheight = 0;
    unsigned int xadder = 0, yadder = 0;
    int geo = 0;                /* return flag from XGetGeometry */
    unsigned int depth, bw=0;
    Window root;
    XSizeHints size_hints;
    XPoint xp;
    XrmValue value;
    size_hints.flags = 0;
    strcpy(progdefaults, "=600x450+0+0");
    if (w) {
        /*
         * The window should be queried for it's size and position. Then the
         * new window should be given almost the same locations
         */
        if (XGetGeometry(gXDisplay, w, &root, &x, &y, &width,
                         &height, &bw, &depth))
        {
            xp = getWindowPositionXY(gXDisplay, w);
            x = xp.x + 40;
            y = xp.y + 40;
            if (x < 0)
                x = 0;
            if (y < 0)
                y = 0;
            size_hints.flags |= (USSize | USPosition);
        }
        else {
            fprintf(stderr,
                 "(HyperDoc) Error Querying window configuration: %ld.\n", w);
            x = y = 0;
            width = 600;
```

```
            height = 450;
            size_hints.flags |= (PSize | PPosition);
        }
    }
    else {
        /* this is the first window, so lets try to find a nice spot for it */
        if (XrmGetResource(rDB, "Axiom.hyperdoc.Geometry",
                                "Axiom.hyperdoc.Geometry",
            str_type, &value) == True)
        {
            strncpy(userdefaults, value.addr, (int) value.size);
            geo = XParseGeometry(userdefaults, &x, &y, &width, &height);
        }
        else
            strcpy(userdefaults, progdefaults);
        size_hints.flags |=(geo & (WidthValue | HeightValue)) ? USSize : PSize;
        size_hints.flags |= (geo & (XValue | YValue)) ? USPosition : PPosition;
        geo = XGeometry(gXDisplay, gXScreenNumber, userdefaults, progdefaults,
                        bw, fwidth, fheight, xadder, yadder,
                        &x, &y, (int *)&width, (int *)&height);
    }
    size_hints.x = x;
    size_hints.y = y;
    size_hints.width = width;
    size_hints.height = height;
    getTitleBarMinimumSize(&(size_hints.min_width), &(size_hints.min_height));
    size_hints.flags |= PMinSize;
    XSetNormalHints(gXDisplay, gWindow->fMainWindow, &size_hints);
    /* just in case a hint isn't enough ... */
    XFlush(gXDisplay);
}
```

9.17.9 getGCs

Create the graphics contexts to be used for all drawing operations.

⟨*hypertex*⟩+≡

```
static void getGCs(HDWindow *window) {
    /*unsigned long valuemask = 0;*/
    XGCValues values;
    values.background = gBackgroundColor;
    window->fStandardGC =
      XCreateGC(gXDisplay, window->fMainWindow, GCBackground, &values);
    XSetLineAttributes(gXDisplay, window->fStandardGC, window->border_width,
                       LineSolid, CapButt, JoinMiter);
    /* create the stipple for the gc */
    stipple = XCreateBitmapFromData(gXDisplay,
        RootWindow(gXDisplay, gXScreenNumber),
        stipple_bits, stipple_width, stipple_height);
    values.background = gInputBackgroundColor;
    values.foreground = gInputForegroundColor;
    values.font = gInputFont->fid;
    if (values.font == server_font )
        window->fInputGC = XCreateGC(gXDisplay, window->fMainWindow,
            GCBackground | GCForeground, &values);
    else {
        window->fInputGC  = XCreateGC(gXDisplay, window->fMainWindow,
            GCBackground | GCForeground | GCFont, &values);
    }
    window->fCursorGC = XCreateGC(gXDisplay, window->fMainWindow, 0, NULL);
    if (values.font != server_font)
        XSetFont(gXDisplay,   window->fCursorGC, gInputFont->fid);
    XSetBackground(gXDisplay, window->fCursorGC, gInputForegroundColor);
    XSetForeground(gXDisplay, window->fCursorGC, gInputBackgroundColor);
    window->fControlGC = XCreateGC(gXDisplay, window->fMainWindow, 0, NULL);
    XSetBackground(gXDisplay, window->fControlGC, gControlBackgroundColor);
    XSetForeground(gXDisplay, window->fControlGC, gControlForegroundColor);
}
```

9.17.10 loadFont

Load a font and store the information in the fontInfo parameter.

⟨*hypertex*⟩+≡

```
static void loadFont(XFontStruct **fontInfo, char *fontname) {
   if ((*fontInfo = XLoadQueryFont(gXDisplay, fontname)) == NULL) {
        fprintf(stderr, "(HyperDoc) Cannot load font %s ; using default.\n",
            fontname);
        if ((*fontInfo = XQueryFont(gXDisplay,
               XGContextFromGC(DefaultGC(gXDisplay, gXScreenNumber)))) == NULL)
        {
            fprintf(stderr, "(HyperDoc) Cannot get default font ; exiting.\n");
            exit(-1);
        }
   }
}
```

9.17.11 ingItColorsAndFonts

This routine initializes all the colors and fonts that the user wishes to use. It checks for all the following properties in `$HOME/.Xdefaults`.

- Axiom.hyperdoc.ActiveColor
- Axiom.hyperdoc.Background
- Axiom.hyperdoc.EmphasizeColor
- Axiom.hyperdoc.EmphasizeFont
- Axiom.hyperdoc.Foreground
- Axiom.hyperdoc.InputBackground
- Axiom.hyperdoc.InputForeground
- Axiom.hyperdoc.SpadColor
- Axiom.hyperdoc.SpadFont

⟨*hypertex*⟩+≡

```
static void ingItColorsAndFonts(void) {
    char property[256];
    char *prop = &property[0];
    char *str_type[50];
    XrmValue value;
    Colormap cmap;
    int ts;
    /** get the color map for the display **/
/*    fprintf(stderr,"initx:ingItColorsAndFonts:entered\n");*/
/*    fprintf(stderr,"initx:ingItColorsAndFonts:DefaultColorMap\n");*/
    cmap = DefaultColormap(gXDisplay, gXScreenNumber);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:initGroupStack\n");*/
    initGroupStack();
    /** then start getting the fonts **/
/*    fprintf(stderr,"initx:ingItColorsAndFonts:mergeDatabases\n");*/
    mergeDatabases();
/*    fprintf(stderr,"initx:ingItColorsAndFonts:XrmGetResource\n");*/
    if (XrmGetResource(rDB, "Axiom.hyperdoc.RmFont",
                           "Axiom.hyperdoc.Font", str_type, &value) == True)
        (void) strncpy(prop, value.addr, (int) value.size);
    else
        (void) strcpy(prop, RmFontDefault);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:loadFont 1\n");*/
    loadFont(&gRmFont, prop);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:loadFont 2\n");*/
```

```
    loadFont(&gInputFont, prop);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:XrmGetResource 2\n");*/
    if (XrmGetResource(rDB, "Axiom.hyperdoc.TtFont",
                           "Axiom.hyperdoc.Font", str_type, &value) == True)
        (void) strncpy(prop, value.addr, (int) value.size);
    else
        (void) strcpy(prop, TtFontDefault);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:loadFont 3\n");*/
    loadFont(&gTtFont, prop);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:isIt850\n");*/
    gTtFontIs850=isIt850(gTtFont);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:XrmGetResource 5\n");*/
    if (XrmGetResource(rDB, "Axiom.hyperdoc.ActiveFont",
                           "Axiom.hyperdoc.Font", str_type, &value) == True)
        (void) strncpy(prop, value.addr, (int) value.size);
    else
        (void) strcpy(prop, ActiveFontDefault);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:loadFont 4\n");*/
    loadFont(&gActiveFont, prop);
    /* maintain backwards compatibility */
/*    fprintf(stderr,"initx:ingItColorsAndFonts:XrmGetResource 6\n");*/
    if (XrmGetResource(rDB, "Axiom.hyperdoc.AxiomFont",
                           "Axiom.hyperdoc.Font", str_type, &value) == True)
        (void) strncpy(prop, value.addr, (int) value.size);
    else {
        if (XrmGetResource(rDB, "Axiom.hyperdoc.SpadFont",
                          "Axiom.hyperdoc.Font", str_type, &value) == True)
        {
            (void) strncpy(prop, value.addr, (int) value.size);
        }
        else {
            (void) strcpy(prop, AxiomFontDefault);
        }
    }
/*    fprintf(stderr,"initx:ingItColorsAndFonts:loadFont 5\n");*/
    loadFont(&gAxiomFont, prop);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:XrmGetResource 7\n");*/
    if (XrmGetResource(rDB, "Axiom.hyperdoc.EmphasizeFont",
                           "Axiom.hyperdoc.Font", str_type, &value) == True)
    {
        (void) strncpy(prop, value.addr, (int) value.size);
    }
    else {
        (void) strcpy(prop, EmphasizeFontDefault);
    }
/*    fprintf(stderr,"initx:ingItColorsAndFonts:loadFont 6\n");*/
```

```
    loadFont(&gEmFont, prop);
/*    fprintf(stderr,"initx:ingItColorsAndFonts:XrmGetResource 8\n");*/
    if (XrmGetResource(rDB, "Axiom.hyperdoc.BoldFont",
                            "Axiom.hyperdoc.Font", str_type, &value) == True)
    {
        (void) strncpy(prop, value.addr, (int) value.size);
    }
    else {
        (void) strcpy(prop, BoldFontDefault);
    }
/*    fprintf(stderr,"initx:ingItColorsAndFonts:loadFont 7\n");*/
    loadFont(&gBfFont, prop);
    /*
     * If we are on a monochrome screen, then we ignore user preferences, and
     * set the foreground and background as I wish
     */
/*    fprintf(stderr,"initx:ingItColorsAndFonts:DisplayPlanes\n");*/
    if (DisplayPlanes(gXDisplay, gXScreenNumber) == 1) {
        gActiveColor       = gAxiomColor
                            = gControlBackgroundColor
                            = gInputBackgroundColor
                            = gBfColor
                            = gEmColor
                            = gRmColor
                            = gSlColor
                            = gTtColor
                            = BlackPixel(gXDisplay, gXScreenNumber);
        gBackgroundColor   = gInputForegroundColor
                            = gControlForegroundColor
                            = WhitePixel(gXDisplay, gXScreenNumber);
    }
    else {

        /*
         * If I have gotten here, then we must be on a color screen, so see
         * what the user likes, and set it up
         */
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 1\n");*/
        gRmColor =
            getColor("RmColor", "Foreground",
                      BlackPixel(gXDisplay, gXScreenNumber), &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 2\n");*/
        gBackgroundColor =
            getColor("Background", "Background",
                      WhitePixel(gXDisplay, gXScreenNumber), &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 3\n");*/
```

```
        gActiveColor =
            getColor("ActiveColor", "Foreground",
                        BlackPixel(gXDisplay, gXScreenNumber), &cmap);
        /*
         * for next two, I want name arg = class arg, ie do not want
         * Background and Foreground.
         */
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 4\n");*/
        gControlBackgroundColor = getColor("ControlBackground",
            "ControlBackground", WhitePixel(gXDisplay, gXScreenNumber), &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 5\n");*/
        gControlForegroundColor = getColor("ControlForeground",
            "ControlForeground", BlackPixel(gXDisplay, gXScreenNumber), &cmap);
        /* maintain backwards compatibility */
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 6\n");*/
        gAxiomColor = getColor("AxiomColor", "Foreground", 0, &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 7\n");*/
        if (gAxiomColor == 0)
            gAxiomColor = getColor("SpadColor", "Foreground",
                BlackPixel(gXDisplay, gXScreenNumber), &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 8\n");*/
        gInputBackgroundColor =
            getColor("InputBackground", "Foreground", gRmColor, &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 9\n");*/
        gInputForegroundColor =
           getColor("InputForeground", "Background", gBackgroundColor, &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 10\n");*/
        gEmColor =
            getColor("EmphasizeColor", "Foreground", gRmColor, &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 11\n");*/
        gTtColor =
            getColor("TtColor", "Foreground", gRmColor, &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 12\n");*/
        gSlColor =
            getColor("EmphasizeColor", "Foreground", gRmColor, &cmap);
/*        fprintf(stderr,"initx:ingItColorsAndFonts:getColor 13\n");*/
        gBfColor =
            getColor("BoldColor", "Foreground", gRmColor, &cmap);
    }
/*    fprintf(stderr,"initx:ingItColorsAndFonts:makeColors\n");*/
    makeColors(gXDisplay, gXScreenNumber, &cmap, &spadColors, &ts);
    /*
     * Now set the current color and font, so I never have to do it again
     */
    gTopOfGroupStack->cur_color = gRmColor;
    gTopOfGroupStack->cur_font = gRmFont;
```

```
/*    fprintf(stderr,"initx:ingItColorsAndFonts:exited\n");*/
}
```

9.17.12 changeText

⟨*hypertex*⟩+≡

```
void changeText(int color, XFontStruct *font) {
    if (font) {
        XGCValues gcv;
        gcv.foreground = color;
        gcv.background = gBackgroundColor;
        XChangeGC(gXDisplay, gWindow->fStandardGC,
                  GCForeground | GCBackground , &gcv);
        if (font->fid != server_font)
            XSetFont(gXDisplay, gWindow->fStandardGC, font->fid);
    }
}
```

9.17.13 getColor

This routine checks the .Xdefaults file of the user for the specified color. If found it allocates a place in the color map for it. If not found, or if an error occurrs, it writes an error message, and uses the given default value.

⟨*hypertex*⟩+≡

```
static int getColor(char *name, char *class, int def, Colormap *map) {
    char fullname[256];
    char fullclass[256];
    char property[256];
    char *prop = &property[0];
    char *str_type[50];
    XrmValue value;
    int ret_val;
    XColor color_def, color_db;
#ifdef DEBUG
    printf("getColor: %s %s %d -> ", name, class, def);
#endif
    strcpy(fullname, "Axiom.hyperdoc.");
    strcat(fullname, name);
    strcpy(fullclass,"Axiom.hyperdoc.");
    strcat(fullclass,class);
    if (XrmGetResource(rDB, fullname, fullclass, str_type, &value) == True) {
        (void) strncpy(prop, value.addr, (int) value.size);
        ret_val=XAllocNamedColor(gXDisplay, *map, prop, &color_def, &color_db);
        if (ret_val) {
#ifdef DEBUG
            printf("%d\n", color_def.pixel);
#endif
            return (color_def.pixel);
        }
        else {
            fprintf(stderr,
                "(HyperDoc) Defaulting on color for %s. Unknown color is %s.\n",
                    name, prop);
#ifdef DEBUG
            printf("%d\n", def);
#endif
            return (def);
        }
    }
    else {
#ifdef DEBUG
        printf("%d\n", def);
#endif
```

```
        return (def);
    }
}
```

9.17.14 mergeDatabases

⟨*hypertex*⟩+≡

```
static void mergeDatabases(void) {
    XrmDatabase homeDB, serverDB, applicationDB;
    char filenamebuf[1024];
    char *filename = &filenamebuf[0];
    char *classname = "Axiom";
    char name[255];
/*    fprintf(stderr,"initx:mergeDatabases:entered\n");*/
/*    fprintf(stderr,"initx:mergeDatabases:XrmInitialize\n");*/
    (void) XrmInitialize();
    (void) strcpy(name, "/usr/lib/X11/app-defaults/");
    (void) strcat(name, classname);
/*  fprintf(stderr,"initx:mergeDatabases:XrmGetFileDatabase name=%s\n",name);*/
    applicationDB = XrmGetFileDatabase(name);
/*    fprintf(stderr,"initx:mergeDatabases:XrmMergeDatabases\n");*/
    (void) XrmMergeDatabases(applicationDB, &rDB);
/*    fprintf(stderr,"initx:mergeDatabases:XrmGetStringDatabase\n");*/
    if (XResourceManagerString(gXDisplay) != NULL) {
        serverDB = XrmGetStringDatabase(XResourceManagerString(gXDisplay));
    }
    else {
        (void) strcpy(filename, getenv("HOME"));
        (void) strcat(filename, "/.Xdefaults");
/*        fprintf(stderr,"initx:mergeDatabases:XrmGetFileDatase\n");*/
        serverDB = XrmGetFileDatabase(filename);
    }
/*    fprintf(stderr,"initx:mergeDatabases:XrmMergeDatabases 2\n");*/
    XrmMergeDatabases(serverDB, &rDB);
    if (getenv("XENVIRONMENT") == NULL) {
        int len;
        (void) strcpy(filename, getenv("HOME"));
        (void) strcat(filename, "/.Xdefaults-");
        len = strlen(filename);
        (void) gethostname(filename + len, 1024 - len);
    }
    else {
        (void) strcpy(filename, getenv("XENVIRONMENT"));
    }
/*    fprintf(stderr,"initx:mergeDatabases:filename=%s\n",filename);*/
    homeDB = XrmGetFileDatabase(filename);
/*    fprintf(stderr,"initx:mergeDatabases:XrmMergeDatabases 3\n");*/
    XrmMergeDatabases(homeDB, &rDB);
}
```

9.17.15 isIt850

⟨*hypertex*⟩+≡

```
int isIt850(XFontStruct *fontarg) {
 char *s;
 int i,val;
 static struct {
      char *name;
      Atom format;
      Atom atom;
      } proptbl = { "CHARSET_ENCODING", XA_ATOM };
 proptbl.atom = XInternAtom(gXDisplay,proptbl.name,0);
 for (i=0;i<fontarg->n_properties;i++)
  {
    if (fontarg->properties[i].name != proptbl.atom) continue;
/* return 1 if it is 850 */
    s = XGetAtomName(gXDisplay,(Atom)fontarg->properties[i].card32);
    val = !( strcmp("850",s) * strcmp("ibm-850",s));
    XFree(s);
    return( val );
  }
 return(0);
}
```

9.18 Handling user page interaction

9.18.1 fillBox

⟨*hypertex*⟩+≡

```
void fillBox(Window w,ImageStruct * image) {
    XClearWindow(gXDisplay, w);
    XPutImage(gXDisplay, w, gWindow->fControlGC,
              image->image.xi, 0, 0, 0, 0,
              image->width,
              image->height);
}
```

9.18.2 toggleInputBox

⟨*hypertex*⟩+≡

```
void toggleInputBox(HyperLink *link) {
    InputBox *box;
    box = link->reference.box;
    if (box->picked) {
        box->picked = 0;
        unpick_box(box);
    }
    else {
        box->picked = 1;
        pick_box(box);
    }
}
```

9.18.3 toggleRadioBox

⟨*hypertex*⟩+≡

```
void toggleRadioBox(HyperLink *link) {
    InputBox *box;
    box = link->reference.box;
    if (box->picked) {
        /*
         * box->picked = 0; unpick_box(box);
         */
    }
    else {
        /* the first thing I do is clear his buddies */
        clearRbs(box->rbs->boxes);
        box->picked = 1;
        pick_box(box);
    }
}
```

9.18.4 clearRbs

⟨*hypertex*⟩+≡

```
static void clearRbs(InputBox *list) {
    InputBox *trace = list;
    while (trace && !trace->picked)
        trace = trace->next;
    if (trace != NULL) {
        trace->picked = 0;
        unpick_box(trace);
    }
}
```

9.18.5 changeInputFocus

⟨*hypertex*⟩+≡

```
void changeInputFocus(HyperLink *link) {
    InputItem *new_item = link->reference.string;
    InputItem *old_item = gWindow->page->currentItem;
    XWindowChanges wc;
    /** first thing I should do is see if the user has clicked in the same
      window that I am in                                          ****/
    if (old_item == new_item)
        return;
    /**  Now change the current pointer **/
    gWindow->page->currentItem = new_item;
    /** Now I have to change the border width of the selected input window **/
    wc.border_width = 1;
    XConfigureWindow(gXDisplay, new_item->win,
                     CWBorderWidth,
                     &wc);
    wc.border_width = 0;
    XConfigureWindow(gXDisplay, new_item->win,
                     CWBorderWidth,
                     &wc);
    updateInputsymbol(old_item);
    updateInputsymbol(new_item);
}
```

9.18.6 nextInputFocus

⟨*hypertex*⟩+≡

```
void nextInputFocus(void) {
    InputItem *old_item = gWindow->page->currentItem, *new_item, *trace;
    if (gWindow->page->currentItem == NULL ||
        (gWindow->page->currentItem->next == NULL
         && gWindow->page->currentItem == gWindow->page->input_list)) {
        BeepAtTheUser();
        return;
    }
    /*
     * Now I should  find the new item
     */
    new_item = NULL;
    trace = old_item->next;
    if (trace == NULL)
        new_item = gWindow->page->input_list;
    else
        new_item = trace;
    gWindow->page->currentItem = new_item;
    drawInputsymbol(old_item);
    drawInputsymbol(new_item);
}
```

9.18.7 prevInputFocus

⟨*hypertex*⟩+≡

```
void prevInputFocus(void) {
    InputItem *old_item = gWindow->page->currentItem, *new_item, *trace;
    if (gWindow->page->currentItem == NULL) {
        BeepAtTheUser();
        return;
    }
    /*
     * Now I should  find the new item
     */
    new_item = NULL;
    trace = gWindow->page->input_list;
    if (trace == old_item) {
        /*
         * I started at the front of the list, so move forward until I hit
         * the end
         */
        while (trace->next != NULL)
            trace = trace->next;
        new_item = trace;
    }
    else {
        while (trace->next != old_item)
            trace = trace->next;
        new_item = trace;
    }
    gWindow->page->currentItem = new_item;
    drawInputsymbol(old_item);
    drawInputsymbol(new_item);
}
```

9.18.8 returnItem

⟨*hypertex*⟩+≡

```
InputItem *returnItem(char *name) {
    InputItem *list;
    list = gWindow->page->input_list;
    while (list != NULL) {
        if (!strcmp(name, list->name))
            return list;
        list = list->next;
    }
    return NULL;
}
```

9.18.9 deleteItem

⟨*hypertex*⟩+≡

```
int deleteItem(char *name) {
    InputItem *list;
    InputItem *prev = NULL;
    list = gWindow->page->input_list;
    while (list != NULL) {
        if (!strcmp(name, list->name)) {
            if (prev)
                prev->next = list->next;
            else
                gWindow->page->input_list = list->next;
            if (gWindow->page->currentItem == list)
                gWindow->page->currentItem = gWindow->page->input_list;
            freeInputItem(list, 1);
            free(list);
            return 1;
        }
        prev = list;
        list = list->next;
    }
    fprintf(stderr, "Can't delete input item %s\n", name);
    return 0;
}
```

9.19 Manipulate the item stack

9.19.1 pushItemStack

⟨*hypertex*⟩+≡

```
void pushItemStack(void) {
    ItemStack *is = (ItemStack *) halloc(sizeof(ItemStack), "Item stack");
    is->indent = indent;
    is->item_indent = item_indent;
    is->next = gTopOfItemStack;
    is->in_item = gInItem;
    gTopOfItemStack = is;
    return;
}
```

9.19.2 clearItemStack

⟨*hypertex*⟩+≡

```
void clearItemStack(void) {
    ItemStack *is = gTopOfItemStack, *chuck;
    while (is != NULL) {
        chuck = is;
        is = is->next;
        free(chuck);
    }
    return;
}
```

9.19.3 popItemStack

⟨*hypertex*⟩+≡

```
void popItemStack(void) {
    ItemStack *chuck;
    if (gTopOfItemStack == NULL) {
        fprintf(stderr, "Tried to pop an empty item stack\n");
        return;
    }
    chuck = gTopOfItemStack;
    gTopOfItemStack = gTopOfItemStack->next;
    indent = chuck->indent;
    item_indent = chuck->item_indent;
    gInItem = chuck->in_item;
    free(chuck);
}
```

9.19.4 copyItemStack

⟨*hypertex*⟩+≡

```
ItemStack *copyItemStack(void) {
    ItemStack *new = NULL;
    ItemStack *prev = NULL;
    ItemStack *trace = gTopOfItemStack;
    ItemStack *first = NULL;
    while (trace) {
        new = (ItemStack *) halloc(sizeof(ItemStack), "Item stack");
        new->indent = trace->indent;
        new->item_indent = trace->item_indent;
        new->in_item = gInItem;
        if (!first)
            first = new;
        else
            prev->next = new;
        prev = new;
        trace = trace->next;
    }
    if (new)
        new->next = NULL;
    return first;
}
```

9.19.5 freeItemStack

⟨*hypertex*⟩+≡

```
void freeItemStack(ItemStack *is) {
    ItemStack *junk = NULL;
    ItemStack *trace = is;
    while (trace) {
        junk = trace;
        trace = trace->next;
        free(junk);
    }
}
```

9.20 Keyboard handling

9.20.1 handleKey

⟨*hypertex*⟩+≡

```
void handleKey(XEvent *event) {
  char key_buffer[20];
  int key_buffer_size = 20;
  KeySym keysym;
  XComposeStatus compstatus;
  int charcount;
  int display_again = 0;
  char *name;
  char *filename;
  /*char *head = "echo htadd -l ";*/
  /*char *blank1 = "                                        ";*/
  /*char *blank2 = "                                       \n";*/
  char buffer[180];
  FILE *filehandle;
  charcount = XLookupString((XKeyEvent *)event, key_buffer, key_buffer_size,
                            &keysym ,&compstatus);
  key_buffer[charcount] = '\0';
  switch (keysym) {
  case XK_Prior:
  case XK_F29:
    scrollUpPage();
    break;
  case XK_Next:
  case XK_F35:
    scrollDownPage();
    break;
  case XK_F3:
  case XK_F12:
    quitHyperDoc();
    break;
  case XK_F5:
    if (event->xkey.state & ShiftMask) {
      name = gWindow->page->name;
      filename = gWindow->page->filename;
      sprintf(buffer, "htadd -l %s\n", filename);
      system(buffer);
      filehandle = (FILE *) hashFind(&gFileHashTable, filename);
      fclose(filehandle);
      hashDelete(&gFileHashTable, filename);
      gWindow->fMacroHashTable =
        (HashTable *) halloc(sizeof(HashTable), "macro hash");
```

```
      hashInit(
                gWindow->fMacroHashTable,
                MacroHashSize,
                (EqualFunction ) stringEqual,
                (HashcodeFunction) stringHash);
      gWindow->fPatchHashTable =
        (HashTable *) halloc(sizeof(HashTable), "patch hash");
      hashInit(
                gWindow->fPatchHashTable,
                PatchHashSize,
                (EqualFunction ) stringEqual,
                (HashcodeFunction) stringHash);
      gWindow->fPasteHashTable =
        (HashTable *) halloc(sizeof(HashTable), "paste hash");
      hashInit(gWindow->fPasteHashTable,
                PasteHashSize,
                (EqualFunction ) stringEqual,
                (HashcodeFunction) stringHash);
      gWindow->fCondHashTable =
        (HashTable *) halloc(sizeof(HashTable), "cond hash");
      hashInit(
                gWindow->fCondHashTable,
                CondHashSize,
                (EqualFunction ) stringEqual,
                (HashcodeFunction) stringHash);
      gWindow->fPageHashTable =
        (HashTable *) halloc(sizeof(HashTable), "page hash");
      hashInit(
                gWindow->fPageHashTable,
                PageHashSize,
                (EqualFunction ) stringEqual,
                (HashcodeFunction) stringHash);
      makeSpecialPages(gWindow->fPageHashTable);
      readHtDb(
                 gWindow->fPageHashTable,
                 gWindow->fMacroHashTable,
                 gWindow->fPatchHashTable);
      gWindow->page = (HyperDocPage *) hashFind(gWindow->fPageHashTable, name);
      if (gWindow->page == NULL) {
        fprintf(stderr, "lose...gWindow->page for %s is null\n", name);
        exit(-1);
      }
      display_again = 1;
    }
    break;
  case XK_F9:
```

```
    makeWindowLink(KeyDefsHelpPage);
    break;
  case XK_Tab:
    if (event->xkey.state & ShiftMask)
      prevInputFocus();
    else if (event->xkey.state & ModifiersMask)
      BeepAtTheUser();
    else
      nextInputFocus();
    break;
  case XK_Return:
    if (!(event->xkey.state & ShiftMask)) {
      nextInputFocus();
      break;
    }
    /* next ones fall through to input area handling */
  case XK_Escape:
    if (!gWindow->page->currentItem)
      break;
  case XK_F1:
    if (!gWindow->page->currentItem) {
      gWindow->page->helppage = allocString(NoMoreHelpPage);
      helpForHyperDoc();
      break;
    }
  case XK_Home:
    if (!gWindow->page->currentItem) {
      scrollToFirstPage();
      break;
    }
  case XK_Up:
    if (!gWindow->page->currentItem) {
      scrollUp();
      break;
    }
  case XK_Down:
    if (!gWindow->page->currentItem) {
      scrollDown();
      break;
    }
  default:
    display_again = 0;
    dialog(event, keysym, key_buffer);
    XFlush(gXDisplay);
    break;
  }
```

```
    if (display_again) {
      displayPage(gWindow->page);
      gWindow->fWindowHashTable = gWindow->page->fLinkHashTable;
    }
  }
```

9.20.2 getModifierMask

This routine returns the modifier mask associated to a key symbol.

⟨*hypertex*⟩+≡

```
  static unsigned int getModifierMask(KeySym sym) {
      unsigned int       i, mask;
      XModifierKeymap    *mod;
      KeyCode            kcode;
      const int          masks[8] = {
          ShiftMask, LockMask, ControlMask,
              Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask
      };
      mod = XGetModifierMapping(gXDisplay);
      kcode = XKeysymToKeycode(gXDisplay,sym);
      if (mod) {
          for (i = 0; i < (8 * mod->max_keypermod); i++){
               if (!mod->modifiermap[i]) continue;
               else if (kcode == mod->modifiermap[i]){
                   mask = masks[i / mod->max_keypermod];
                   XFreeModifiermap(mod);
                   return mask;
               }
          }
          XFreeModifiermap(mod);
      }
      return 0;
  }
```

9.20.3 initKeyin

This routine initializes some of the variables needed by the input strings, and boxes.

⟨*hypertex*⟩+≡

```
void initKeyin(void) {
    char *prop;
    unsigned int nlm;
    nlm = getModifierMask(XK_Num_Lock);
    UnsupportedModMask &= ~nlm;
    ModifiersMask &= ~nlm;
    /*
     * First set all the values for when the active cursor is in the window
     */
    in_cursor_height = 2;
    in_cursor_y = gInputFont->max_bounds.ascent +
        gInputFont->max_bounds.descent;
    in_cursor_width = gInputFont->max_bounds.width;
    /*
     * Now for when the cursor is empty
     */
    out_cursor_height = gInputFont->max_bounds.ascent +
        gInputFont->max_bounds.descent;
    out_cursor_y = 2;
    out_cursor_width = in_cursor_width;
    start_x = 5;
    start_y = gInputFont->max_bounds.ascent;
    /*
     * Find out How big I should make the simple boxes
     */
    simple_box_width = XTextWidth(gInputFont, "X", 1) + 5;
    prop = XGetDefault(gXDisplay, gArgv[0], "ProtectedQuit");
    if (prop == NULL) {
        protected_quit = (char *) halloc(strlen("ProtectedPage") + 1,
                                         "protected_quit");
        strcpy(protected_quit, "ProtectedPage");
    }
    else {
        protected_quit = (char *) halloc(strlen(prop) + 1, "protected_quit");
        strcpy(protected_quit, prop);
    }
}
```

9.21 Handle page macros

9.21.1 scanHyperDoc

This routine keeps scanning until it reaches it pops off 1 more right brace then left brace.

⟨*hypertex*⟩+≡

```
void scanHyperDoc(void) {
    HDWindow *twin = gWindow;
    int ret_val;
    int number_of_left_braces = 1;
    gWindow = NULL;
    while (number_of_left_braces) {
        ret_val = getToken();
        if (ret_val == EOF && number_of_left_braces) {
            fprintf(stderr, "Scan_Hypertex: Unexpected End of File\n");
            longjmp(jmpbuf, 1);
        }
        switch (token.type) {
          case Page:
            fprintf(stderr, "scanHyperDoc: Unexpected Page Declaration\n");
            break;
          case NewCommand:
            fprintf(stderr, "scanHyperDoc: Unexpected Macro Declaration\n");
            break;
          case Lbrace:
            number_of_left_braces++;
            break;
          case Endpatch:
          case Rbrace:
            number_of_left_braces--;
            break;
          default:
            break;
        }
    }
    gWindow = twin;
}
```

9.21.2 number

⟨*hypertex*⟩+≡

```
int number(char *str) {
    char *t = str;
    while (*t)
        if (!isdigit(*t++))
            return 0;
    return 1;
}
```

9.21.3 loadMacro

Parse a given macro given the pointer to the unloaded macro.

⟨*hypertex*⟩+≡

```
static char *loadMacro(MacroStore *macro) {
    int ret_val;
    long start_fpos;
    int size = 0;
    char *trace;
    char *macro_buff;
    saveScannerState();
    cfile = findFp(macro->fpos);
    initScanner();
    /** First thing I should do is make sure that the name is correct ***/
    getExpectedToken(NewCommand);
    getExpectedToken(Lbrace);
    getExpectedToken(Macro);
    if (strcmp(token.id, macro->name)) {
        /** WOW, Somehow I had the location of the wrong macro **/
        fprintf(stderr, "Expected macro name %s got insted %s in loadMacro\n",
                macro->name, token.id);
        longjmp(jmpbuf, 1);
    }
    getExpectedToken(Rbrace);
    /** Next I should check to see if I have any parameters **/
    getToken();
    if (token.type == Lsquarebrace) {
       /** The person is telling me the number of macros he is going to use **/
        getExpectedToken(Word);
        if (!number(token.id)) {
            fprintf(stderr, "loadMacro: Expected A Value Instead Got %s\n",
                    token.id);
            longjmp(jmpbuf, 1);
        }
        /** if it is a number, then I should store it in the parameter number
          member of the macro structure **/
        macro->number_parameters = atoi(token.id);
#ifdef DEBUG
        fprintf(stderr,
              "The number of parameters is %d\n", macro->number_parameters);
#endif
        getExpectedToken(Rsquarebrace);
        getToken();
    }
    else
```

```
        macro->number_parameters = 0;
    /*** Now I should be able to check the token, and insure that I have read
      a leftbrace, then the string will follow                    ****/
    if (token.type != Lbrace) {
        /** The macro is not in a group, uh oh **/
        fprintf(stderr, "loadMacro:Expected a Left Brace got type %d\n",
                token.type);
        longjmp(jmpbuf, 1);
    }
    start_fpos = fpos;
    scanHyperDoc();
    ret_val = fseek(cfile, macro->fpos.pos + start_fpos, 0);
    size = fpos - start_fpos;
    macro_buff = (char *) halloc((size + 1) * sizeof(char), "Macro_buf");
    for (size = 0, trace = macro_buff; size < fpos - (start_fpos) - 1; size++)
        *trace++ = getc(cfile);
    *trace = '\0';
    macro->loaded = 1;
    restoreScannerState();
    return macro_buff;
}
```

9.21.4 initParameterElem

⟨*hypertex*⟩+≡

```
ParameterList initParameterElem(int number) {
    ParameterList new;
    int count;
    /** allocate the space neeeded **/
    new = (ParameterList) halloc(sizeof(struct parameter_list_type),
                                 "ParameterList");
    /** now allocate the memeory  for the pointers to the  parameters **/
    if (number) {
       new->list = (char **) halloc(number * sizeof(char *), "Parameter List");
        /** initialize my pointers **/
        for (count = 0; count < number; count++)
            (new->list)[count] = NULL;
    }
    new->number = number;
    return new;
}
```

9.21.5 pushParameters

⟨*hypertex*⟩+≡

```
int pushParameters(ParameterList new) {
    if (new == NULL) {
       fprintf(stderr, "Tried pushing a null list onto the parameter stack\n");
       longjmp(jmpbuf, 1);
    }
    new->next = parameters;
    parameters = new;
    return 1;
}
```

9.21.6 popParameters

Simply pops the top of the parameter list, being good and freeing all the memory.

⟨*hypertex*⟩+≡

```
int popParameters(void) {
    ParameterList old;
    int count;
    if (!parameters) {
        return 0;
    }
    old = parameters;
    parameters = old->next;
    /** Free the parameter text and pointers **/
    if (old->number >0) {
        for (count = 0; count < old->number; count++)
          if ( (old->list)[count] )  free((char *) (old->list)[count]);
        free(old->list);
    }
    free(old);                  /** free the parameter **/
    return 1;
}
```

9.21.7 parseMacro

This routine loads a macro if needed, and then parses it from the string.

⟨*hypertex*⟩+≡

```
int parseMacro(void) {
    MacroStore *macro;
    int s;
    curr_node->type = Macro;
    curr_node->space = token.id[-1];
    curr_node->next = allocNode();
    curr_node = curr_node->next;
    macro = (MacroStore *) hashFind(gWindow->fMacroHashTable, token.id);
    if (macro != NULL) {
        if (!macro->loaded)
            macro->macro_string = loadMacro(macro);
        getParameterStrings(macro->number_parameters, macro->name);
        parseFromString(macro->macro_string);
        if (gEndedPage) {
            s = curr_node->type;
            curr_node->type = Endmacro;
            curr_node->next = allocNode();
            curr_node = curr_node->next;
            curr_node->type = s;
        }
        else
            curr_node->type = Endmacro;
        if (popParameters())
            return 1;
        else {
            fprintf(stderr,
                    "parseMacro: Tried to pop an empty paramter stack\n");
            longjmp(jmpbuf, 1);
        }
    }
    else {
        fprintf(stderr, "parseMacro: Unknown keyword %s\n", token.id);
        longjmp(jmpbuf, 1);
    }
}
```

9.21.8 getParameterStrings

⟨*hypertex*⟩+≡

```
static void getParameterStrings(int number,char * macro_name) {
    static char buffer[4096];
    char *buffer_pntr;
    int count;
    int lbrace_counter;
    char c;
    int size;
    ParameterList new = initParameterElem(number);
    int pnum;
    char pnum_chars[5];
    int pc;
    if (!number) {              /* nothing to be done */
        pushParameters(new);
        return;
    }
    for (count = 0; count < number; count++) {
        getToken();
        if (token.type != Lbrace) {
            /** The macro is not in a group, uh oh **/
            fprintf(stderr, "Wrong number of arguments to the macro %s\n",
                    macro_name);
            jump();
        }
        for (lbrace_counter = 1, buffer_pntr = buffer;
             lbrace_counter;) {
            switch (c = getChar()) {
              case EOF:
                fprintf(stderr, "GetParameterStrings: Unexpected EOF\n");
                longjmp(jmpbuf, 1);
              case '}':
                lbrace_counter--;
                if (lbrace_counter)
                    *buffer_pntr++ = c;
                break;
              case '{':
                lbrace_counter++;
                *buffer_pntr++ = c;
                break;
              case '#':
                /* uh oh, I have a paramter reference inside a paramter */
                /* get the number */
                if (parameters == NULL) {
                    *buffer_pntr++ = c;
```

```
                    break;
                }
                if (
                    ((buffer_pntr > buffer + 1) &&
                     *(buffer_pntr - 1) == '\\' &&
                     *(buffer_pntr - 2) != '\\') ||
                    ((buffer_pntr > buffer) &&
                     *(buffer_pntr - 1) == '\\')) {
                    /* I had a \# */
                    *buffer_pntr++ = c;
                }
                else {
                    c = getChar();
                    for (pc = 0; numeric(c); pc++) {
                        pnum_chars[pc] = c;
                        c = getChar();
                    }
                    ungetChar(c);
                    pnum_chars[pc] = '\0';
                    pnum = atoi(pnum_chars);
                    pc = 0;
                    /* Now copy the paramter */
                    while ((parameters->list)[pnum - 1][pc] != '\0')
                        *buffer_pntr++ = (parameters->list)[pnum - 1][pc++];
                }
                break;
              default:
                *buffer_pntr++ = c;
                break;
            }
        }
        *buffer_pntr = '\0';
        /*** Now add it to the current parameter list **/
        size = strlen(buffer) + 1;
        new->list[count] = (char *) halloc(size, "Parameter Strings");
        strcpy(new->list[count], buffer);
    }
    pushParameters(new);
    return ;
}
```

9.21.9 parseParameters

⟨*hypertex*⟩+≡

```
void parseParameters(void) {
    int value;
    if (!number(token.id)) {
        fprintf(stderr,
                "Parse_parameter: Error Expected a number, got %s instead\n",
                token.id);
        longjmp(jmpbuf, 1);
    }
    if ((value = atoi(token.id)) > parameters->number) {
        /** had a bad parameter number **/
        fprintf(stderr,
                "Parse_parameter: Had a bad parameter number %d\n", value);
        longjmp(jmpbuf, 1);
    }
    parseFromString((parameters->list)[value - 1]);
    curr_node->type = Endparameter;
    return;
}
```

9.22 Memory management routines

9.22.1 freeIfNonNULL

⟨*hypertex*⟩+≡

```
static void freeIfNonNULL(void *p) {
  if (p){
    free(p);
  }
}
```

9.22.2 allocHdWindow

Allocate an HDWindow Structure and initialize it.

⟨*hypertex*⟩+≡

```
HDWindow *allocHdWindow(void) {
  HDWindow *w = (HDWindow *) halloc(sizeof(HDWindow), "HDWindow");
  w->fMemoStack = (HyperDocPage **)
    halloc(MaxMemoDepth * sizeof(HyperDocPage *), "Memo Stack");
  w->fDownLinkStack = (HyperDocPage **)
    halloc(MaxDownlinkDepth * sizeof(HyperDocPage *), "downlink stack");
  w->fDownLinkStackTop =
    (int *) halloc(MaxDownlinkDepth * sizeof(int), "top downlink stack");
  w->fAxiomFrame = 0;
  initPageStructs(w);
  /* Now I initialize the hash tables for the page */
  w->fCondHashTable = (HashTable *) halloc(sizeof(HashTable), "cond hash");
  hashInit(
            w->fCondHashTable,
            CondHashSize,
            (EqualFunction) stringEqual,
            (HashcodeFunction) stringHash);
  w->fPasteHashTable = (HashTable *) halloc(sizeof(HashTable), "paste hash");
  hashInit(
            w->fPasteHashTable,
            PasteHashSize,
            (EqualFunction) stringEqual,
            (HashcodeFunction) stringHash);
  w->fPageHashTable = hashCopyTable(&init_page_hash);
  w->fPatchHashTable = hashCopyTable(&init_patch_hash);
  w->fMacroHashTable = hashCopyTable(&init_macro_hash);
  gWindow = w;
  makeSpecialPages(w->fPageHashTable);
  w->fDisplayedCursor = 0;
  return w;
}
```

9.22.3 freeHdWindow

⟨*hypertex*⟩+≡

```
void freeHdWindow(HDWindow *w) {
  if (w) {
    free(w->fMemoStack);
    free(w->fDownLinkStack);
    free(w->fDownLinkStackTop);
    /*
      free(w->fWindowHashTable); will be taken care of by freeing
      freeHash(w->fPageHashTable, freePage); below
      cf freePage
      */
    freeHash(w->fMacroHashTable, (FreeFunction)dontFree);
    freeHash(w->fPasteHashTable, (FreeFunction)dontFree);
    freeHash(w->fPatchHashTable, (FreeFunction)dontFree);
    freeHash(w->fCondHashTable, (FreeFunction)freeCond);
    freeHash(w->fPageHashTable, (FreeFunction)freePage);
    free(w->fPageHashTable);
    free(w->fPatchHashTable);
    free(w->fMacroHashTable);
    XFreeGC(gXDisplay, w->fStandardGC);
    XFreeGC(gXDisplay, w->fInputGC);
    XFreeGC(gXDisplay, w->fCursorGC);
    XFreeGC(gXDisplay, w->fControlGC);
    free(w);
  }
}
```

9.22.4 allocNode

Allocate an empty text node.

⟨*hypertex*⟩+≡

```
TextNode *allocNode(void) {
  TextNode *temp_node;
  temp_node = (TextNode *) halloc(sizeof(TextNode), "Text Node");
  temp_node->type = 0;
  temp_node->space = 0;
  temp_node->height = 0;
  temp_node->width = 0;
  temp_node->x = -1;
  temp_node->y = -1;
  temp_node->data.node = NULL;
  temp_node->next = NULL;
  temp_node->link = NULL;
  temp_node->image.pm = 0;
  return temp_node;
}
```

9.22.5 freeNode

⟨*hypertex*⟩+≡

```
void freeNode(TextNode *node, short int des) {
  if (node == NULL)
    return;
  switch (node->type) {
  case Paste:
    freePastearea(node, des);
    freeNode(node->next, des);
    break;
  case Pastebutton:
    freePastebutton(node, des);
    freeNode(node->next, des);
    break;
  case Ifcond:
    freeNode(node->data.ifnode->cond, des);
    freeNode(node->data.ifnode->thennode, des);
    freeNode(node->data.ifnode->elsenode, des);
    break;
  case Dash:
  case Lsquarebrace:
  case Word:
  case WindowId:
  case Punctuation:
  case Lbrace:
  case Rbrace:
  case SimpleBox:
  case Verbatim:
  case Math:
  case Spadsrctxt:
  case Spadsrc:
    freeIfNonNULL(node->data.text);
    freeNode(node->next, des);
    break;
  case Inputstring:
    if (des)
      deleteItem(node->data.text);
    freeIfNonNULL(node->data.text);
    freeNode(node->next, des);
    break;
  case It:
  case Sl:
  case Tt:
  case Rm:
  case Emphasize:
```

```
case Beep:
case BoldFace:
case Par:
case Newline:
case Horizontalline:
case Item:
case Beginscroll:
case Endscroll:
case Group:
case Table:
case Macro:
case Pound:
case Center:
case Box:
case Mbox:
case Tableitem:
case Scrollingnode:
case Headernode:
case Titlenode:
case Footernode:
case Controlbitmap:
case Fi:
case Description:
case Rsquarebrace:
case Endpaste:
case Endpastebutton:
  freeNode(node->next, des);
  break;
case Inputbitmap:
case Inputpixmap:
  freeIfNonNULL(node->data.text);
  freeNode(node->next, des);
  break;
case Quitbutton:
case Helpbutton:
case Upbutton:
case Returnbutton:
  if (des && node->link->win) {
    hashDelete(gWindow->page->fLinkHashTable,(char *) &node->link->win);
    XDestroyWindow(gXDisplay, node->link->win);
  }
  freeIfNonNULL(node->link);
  freeNode(node->next, des);
  break;
case Memolink:
case Downlink:
```

```
case Windowlink:
case Link:
case Lisplink:
case Lispwindowlink:
case Spadcall:
case Spadcallquit:
case LispMemoLink:
case Lispcommand:
case Lispcommandquit:
case LispDownLink:
case Unixlink:
case Spadlink:
case Spadmemolink:
case Spaddownlink:
case Unixcommand:
case Spadcommand:
case Spadgraph:
  if (des && node->link->win) {
    hashDelete(gWindow->page->fLinkHashTable,(char *) &node->link->win);
    XDestroyWindow(gXDisplay, node->link->win);
  }
  /* TTT don't free the link before freeing nodes off it */
  /* freeNode(node->link->reference.node);*/
  freeIfNonNULL(node->link);
  freeNode(node->next, des);
  break;
case Free:
case Indent:
case Indentrel:
case HSpace:
case Space:
case VSpace:
case Button:
case Bound:
case Tab:
  freeNode(node->next, des);
  freeNode(node->data.node, des);
  break;
case End:
case Endcenter:
case Endlink:
case Endgroup:
case Endbox:
case Endmbox:
case Endspadcommand:
case Endpix:
```

```
    case Endmacro:
    case Endparameter:
    case Endtable:
    case Endtableitem:
    case Noop:
    case Endinputbox:
    case Enddescription:
    case Endif:
    case Endtitems:
    case Enditems:
    case Endverbatim:
    case Endmath:
    case Endspadsrc:
      freeNode(node->next, des);
      break;
    case Endheader:
    case Endtitle:
    case Endfooter:
    case Endscrolling:
    case Endarg:
      break;
    case Endbutton:
    case Beginitems:
      freeIfNonNULL(node->data.text);
      freeNode(node->next, des);
      break;
    default:
      /*        printf("don't know how to free type %d\n", node->type); */
      return;
    }
    free(node);
  }
```

9.22.6 allocIfnode

⟨*hypertex*⟩+≡

```
  IfNode *allocIfnode(void) {
    IfNode *tempif;
    tempif = (IfNode *) halloc(sizeof(struct if_node), "IfNode");
    tempif->thennode = tempif->elsenode = tempif->cond = NULL;
    return tempif;
  }
```

9.22.7 allocCondnode

⟨*hypertex*⟩+≡

```
CondNode *allocCondnode(void) {
  CondNode *temp;
  temp = (CondNode *) halloc(sizeof(struct cond_node), "Cond Node");
  temp->cond = temp->label = NULL;
  return temp;
}
```

9.22.8 freeCond

⟨*hypertex*⟩+≡

```
static void freeCond(CondNode *cond) {
  if (cond) {
    free(cond->label);
    if (cond->cond)
      free(cond->cond);
    free(cond);
  }
}
```

9.22.9 allocPage

Allocate a new HyperDoc page.

⟨*hypertex*⟩+≡

```
HyperDocPage *allocPage(char *name) {
  HyperDocPage *page;
  page = (HyperDocPage *) halloc(sizeof(HyperDocPage), "HyperDocPage");
  page->name = name;
  page->header = page->scrolling = page->footer = page->title = NULL;
  page->scroll_off = 0;
  page->sock = NULL;
  page->box_hash = page->depend_hash = NULL;
  page->fLinkHashTable =
    (HashTable *) halloc(sizeof(HashTable), "Page->fLinkHashTable");
  page->input_list = page->currentItem = NULL;
  page->pageFlags = 0000000;
  page->filename = NULL;
  page->helppage = allocString(TopLevelHelpPage);
  page->radio_boxes = NULL;
  page->button_list = NULL;
  page->s_button_list = NULL;
  return page;
}
```

9.22.10 freePage

This routine now checks for an environment variable NOFREE. If found it returns. At least, that's what the comment claims but I see no code to implement this. It's not a bad idea though.

⟨*hypertex*⟩+≡

```
void freePage(HyperDocPage *page) {
  if (page == NULL)
    return;
  switch (page->type) {
  case UlUnknownPage:
  case UnknownPage:
  case ErrorPage:
  case Unixfd:
  case SpadGen:
  case Normal:
    /*
     * if(page->name) free(page->name); if(page->filename)
     * free(page->filename);
     */
    freeNode(page->scrolling, 0);
    freeNode(page->header, 0);
    freeNode(page->footer, 0);
    freeNode(page->title, 0);
    freeButtonList(page->s_button_list);
    freeButtonList(page->button_list);
/*
     if (page->sock != NULL)
      free(page->sock);
*/
    freeHash(page->depend_hash, (FreeFunction)freeDepend);
    /* TTT line below causes freeing of freed memory and freed memory reads
       links should have been freed by the recursive freeNode's above
       (cf.freeNode)
       this is apparently because we are called from freeHdWindow
       and we had made a call to free w->fWindowHashTable which is made
       to point to the same thing so we do it HERE not THERE
       */
    freeHash(page->fLinkHashTable, (FreeFunction)dontFree);
    freeHash(page->box_hash, (FreeFunction)freeInputBox);
    freeInputList(page->input_list);
    freeRadioBoxes(page->radio_boxes);
    free(page->helppage);
    free(page);
    break;
```

```
    case UnloadedPageType:
      break;
    default:
      /* fprintf(stderr, "Unknown Page type: %d\n", page->type); */
      break;
    }
  }
```

9.22.11 freePaste

⟨*hypertex*⟩+≡

```
  static void freePaste(PasteNode *paste, short int des) {
    if (paste) {
      freeGroupStack(paste->group);
      freeItemStack(paste->item_stack);
      freeNode(paste->arg_node, des);
      free(paste);
    }
  }
```

9.22.12 freePastebutton

⟨*hypertex*⟩+≡

```
static void freePastebutton(TextNode *node, short int des) {
  /*
   * if I am freeing from within parse patch, then I have to do some
   * special things first
   */
  /* the following seems to be unused */
  if (gActiveWindow == node->link->win)
    gActiveWindow = -1;
  if (des) {
    PasteNode *paste;
    paste = (PasteNode *) hashFind(gWindow->fPasteHashTable, node->data.text);
    if (!paste->haspaste) {
      /* squash this thing */
      hashDelete(gWindow->fPasteHashTable, (char *)node->data.text);
      freePaste(paste, des);
      hashDelete(gWindow->page->fLinkHashTable,(char *) &node->link->win);
      XDestroyWindow(gXDisplay, node->link->win);
    }
    else
      paste->hasbutton = 0;
  }
  freeIfNonNULL(node->data.text);
}
```

9.22.13 freePastearea

⟨*hypertex*⟩+≡
```
static void freePastearea(TextNode *node, short int des) {
  if (des) {
    PasteNode *paste;
    paste = (PasteNode *) hashFind(gWindow->fPasteHashTable, node->data.text);
    if (paste) {
      if (!paste->hasbutton) {
        /* squash this thing */
        hashDelete(gWindow->fPasteHashTable, node->data.text);
        freePaste(paste, des);
      }
      else
        paste->haspaste = 0;
    }
  }
  freeIfNonNULL(node->data.text);
}
```

9.22.14 freeString

⟨*hypertex*⟩+≡
```
void freeString(char *str) {
  freeIfNonNULL(str);
}
```

9.22.15 freeDepend

⟨*hypertex*⟩+≡
```
static void freeDepend(SpadcomDepend *sd) {
  freeIfNonNULL((char *) sd);
}
```

9.22.16 dontFree

⟨*hypertex*⟩+≡
```
static void dontFree(void  *link) {
  return;
}
```

9.22.17 freeLines

⟨*hypertex*⟩+≡

```
static void freeLines(LineStruct *lines) {
  if (lines->prev != NULL)
    lines->prev->next = NULL;
  while (lines != NULL) {
    LineStruct *del;
    del = lines;
    lines = lines->next;
    free(del->buffer);
    free(del);
  }
}
```

9.22.18 freeInputItem

⟨*hypertex*⟩+≡

```
void freeInputItem(InputItem *sym, short int des) {
  freeIfNonNULL(sym->name);
  freeLines(sym->lines);
  if (des)
    XDestroyWindow(gXDisplay, sym->win);
}
```

9.22.19 freeInputList

⟨*hypertex*⟩+≡

```
void freeInputList(InputItem *il) {
  while (il) {
    InputItem *trash = il;
    il = il->next;
    freeInputItem(trash, 0);
    free(trash);
  }
}
```

9.22.20 freeInputBox

⟨*hypertex*⟩+≡
```
static void freeInputBox(InputBox *box) {
  if (box) {
    freeIfNonNULL(box->name);
    free(box);
  }
}
```

9.22.21 freeRadioBoxes

⟨*hypertex*⟩+≡
```
static void freeRadioBoxes(RadioBoxes *radio) {
  if (radio) {
    freeRadioBoxes(radio->next);
    freeIfNonNULL(radio->name);
    free(radio);
  }
}
```

9.22.22 allocInputline

⟨*hypertex*⟩+≡
```
LineStruct *allocInputline(int size) {
  int i;
  LineStruct *line =
    (LineStruct *) halloc(sizeof(LineStruct), "Line Structure");
  line->prev = line->next = NULL;
  line->buffer = (char *) halloc(sizeof(char) * size + 2, "symbol buffer");
  for (i = 0; i < size + 2; i++)
    line->buffer[i] = 0;
  line->buff_pntr = line->len = 0;
  return line;
}
```

9.22.23 allocPasteNode

⟨*hypertex*⟩+≡

```
PasteNode *allocPasteNode(char *name) {
  PasteNode *pastenode =
    (PasteNode *) halloc(sizeof(PasteNode), "PasteNode");
  pastenode->group = NULL;
  pastenode->item_stack = NULL;
  pastenode->arg_node = NULL;
  pastenode->end_node = NULL;
  pastenode->name = allocString(name);
  pastenode->haspaste = pastenode->hasbutton = 0;
  return pastenode;
}
```

9.22.24 allocPatchstore

⟨*hypertex*⟩+≡

```
PatchStore *allocPatchstore(void) {
  PatchStore *p = (PatchStore *) halloc(sizeof(PatchStore), "PatchStore");
  p->loaded = 0;
  p->string = NULL;
  return p;
}
```

9.22.25 freePatch

⟨*hypertex*⟩+≡

```
void freePatch(PatchStore *p) {
  if (p) {
    if (p->name)
      free(p->name);
    if (p->fpos.name)
      free(p->fpos.name);
    if (p->string)
      free(p->string);
    free(p);
  }
}
```

9.22.26 allocInputbox

⟨*hypertex*⟩+≡

```
InputBox *allocInputbox(void) {
  InputBox *box = (InputBox *) halloc(sizeof(InputBox), "InputBox");
  box->picked = 0;
  box->next = NULL;
  box->rbs = NULL;
  return box;
}
```

9.22.27 allocRbs

⟨*hypertex*⟩+≡

```
RadioBoxes *allocRbs(void) {
  RadioBoxes *newrb = (RadioBoxes *) halloc(sizeof(RadioBoxes), "Radio Boxes");
  newrb->next = NULL;
  newrb->boxes = NULL;
  return newrb;
}
```

9.22.28 allocButtonList

⟨*hypertex*⟩+≡

```
ButtonList *allocButtonList(void) {
  ButtonList *newbl = (ButtonList *) halloc(sizeof(ButtonList), "Button List");
  newbl->link = NULL;
  newbl->x0 = newbl->y0 = newbl->x1 = newbl->y1 = 0;
  newbl->next = NULL;
  return newbl;
}
```

9.22.29 freeButtonList

⟨*hypertex*⟩+≡

```
void freeButtonList(ButtonList *bl) {
  while (bl) {
    ButtonList *nbl = bl->next;
    free(bl);
    bl = nbl;
  }
}
```

9.22.30 resizeBuffer

Resizable static buffers.

⟨*hypertex*⟩+≡

```
char *resizeBuffer(int size, char *oldBuf, int *oldSize) {
  char *newBuf;
  int newSize;
  if (size <= *oldSize)
    return oldBuf;
  newSize = size + BufferSlop;
  newBuf = (char *) halloc(newSize,"Buffer");
  memset(newBuf,'\0',newSize);
  if (oldBuf) {
    memcpy(newBuf, oldBuf, *oldSize);
    free(oldBuf);
  }
  *oldSize = newSize;
  return newBuf;
}
```

9.23 Page parsing routines

9.23.1 PushMR

⟨*hypertex*⟩+≡

```
static void PushMR(void) {
    MR_Stack *newStackItem =
       (MR_Stack *) halloc(sizeof(MR_Stack), "Mode Region Stack");
    newStackItem->fParserMode = gParserMode;
    newStackItem->fParserRegion = gParserRegion;
    newStackItem->fNext = top_mr_stack;
    top_mr_stack = newStackItem;
}
```

9.23.2 PopMR

⟨*hypertex*⟩+≡

```
static void PopMR(void) {
    MR_Stack *old = top_mr_stack;
    if (old == NULL) {
        fprintf(stderr,
                   "(HyperDoc) Parser Error: Tried to pop empty MR Stack\n");
        exit(-1);
    }
    else {
        gParserMode = old->fParserMode;
        gParserRegion = old->fParserRegion;
        top_mr_stack = old->fNext;
        free(old);
    }
}
```

9.23.3 loadPage

⟨*hypertex*⟩+≡

```
void loadPage(HyperDocPage *page) {
    if (page->type == UnloadedPageType) {
        HyperDocPage *new_page;
        initScanner();
        new_page = formatPage((UnloadedPage *)page);
        gWindow->page = new_page;
        /* free(page); */
        page = new_page;
    }
}
```

9.23.4 displayPage

Display a HyperDoc page with the given name, parsing it if needed.

⟨*hypertex*⟩+≡

```
void displayPage(HyperDocPage *page) {
    HyperDocPage *new_page;
    XUnmapSubwindows(gXDisplay, gWindow->fMainWindow);
    XUnmapSubwindows(gXDisplay, gWindow->fScrollWindow);
    XFlush(gXDisplay);
    if (setjmp(jmpbuf)) {
        /*
         * since I did not finish formatting the page, let me get rid of what
         * I had
         */
        freePage(formatpage);
        /* Replace the buggy page with what I started with */
        hashReplace(gWindow->fPageHashTable, (char *)page, formatpage->name);
        if (!strcmp(formatpage->name, "ErrorPage")) {
            fprintf(stderr, "(HyperDoc) Oops the error page is buggy\n");
            exit(-1);
        }
        gWindow->page = page =
           (HyperDocPage *) hashFind(gWindow->fPageHashTable, "ErrorPage");
        if (page == NULL) {
            fprintf(stderr, "(HyperDoc) No error page found, exiting\n");
            exit(-1);
        }
        resetConnection();
    }
    if (page->type == UnloadedPageType || page->type == ErrorPage) {
        /* Gack! (page should be a union!) */
        initScanner();
        new_page = formatPage((UnloadedPage *)page);
        gWindow->page = new_page;
        /* free(page); */
        page = new_page;
    }
    showPage(page);
}
```

9.23.5 formatPage

Parse a given HyperDoc Page, from the top.

⟨*hypertex*⟩+≡

```
static HyperDocPage *formatPage(UnloadedPage *ulpage) {
    /*int ret_val;*/
    HyperDocPage *page = allocPage(ulpage->name);
    /*
     * In case of an error I will have to get at this page so I can free the
     * waisted memory
     */
    formatpage = page;
    page->type = Normal;
    hashReplace(gWindow->fPageHashTable, (char *)page, ulpage->name);
    cfile = findFp(ulpage->fpos);
    page->filename = allocString(ulpage->fpos.name);
    parsePage(page);
    return page;
}

/* parse the HyperDoc statements in the given string */
```

9.23.6 parseFromString

⟨*hypertex*⟩+≡

```
void parseFromString(char *str) {
    saveScannerState();
    last_ch = NoChar;
    last_token = 0;
    inputString = str;
    inputType = FromString;
    parseHyperDoc();
    restoreScannerState();
}
```

9.23.7 parseTitle

⟨*hypertex*⟩+≡

```
static void parseTitle(HyperDocPage *page) {
    TextNode *node;
    PushMR();
    gParserRegion = Title;
    getExpectedToken(Lbrace);
    node = allocNode();
    page->title = node;
    node->type = Titlenode;
    node->next = allocNode();
    node = node->next;
    node->type = Center;
    node->next = allocNode();
    curr_node = node->next;
    parseHyperDoc();
    curr_node->type = Endcenter;
    curr_node->next = allocNode();
    curr_node = curr_node->next;
    curr_node->type = Endtitle;
    curr_node->next = NULL;
    if (gNeedIconName) {
        char *title = printToString(page->title);
        XSetIconName(gXDisplay, gWindow->fMainWindow, title);
        gNeedIconName = 0;
    }
    if (token.type != Rbrace) {
      fprintf(stderr,"(HyperDoc) Parse title was expecting a closing brace\n");
      printPageAndFilename();
      jump();
    }
    linkTitleBarWindows();
    PopMR();
}
```

9.23.8 parseHeader

⟨*hypertex*⟩+≡

```
static void parseHeader(HyperDocPage *page) {
    TextNode *node;
    PushMR();
    gParserRegion = Header;
    node = allocNode();
    page->header = node;
    node->type = Headernode;
    node->next = allocNode();
    curr_node = node->next;
    parseHyperDoc();
}

/*
 * parse a page from the top level
 */
```

9.23.9 initParsePage

Parse a page from the top level.

⟨*hypertex*⟩+≡

```
static void initParsePage(HyperDocPage *page) {
    gEndedPage = gInDesc = gStringValueOk = gInIf =
        gInButton = gInOptional = gInVerbatim = gInPaste = gInItems =
        gInSpadsrc = FALSE;
    example_number = 1;
    cur_page = page;
    gParserMode = AllMode;
    /* Now I should set the input list to be null */
    freeInputList(page->input_list);
    page->input_list = page->currentItem = NULL;
    initTopGroup();
    clearBeStack();
    cur_spadcom = NULL;
    gLinkHashTable = page->fLinkHashTable;
    hashInit(
              gLinkHashTable,
              LinkHashSize,
              (EqualFunction) windowEqual,
              (HashcodeFunction) windowCode);
    gPageBeingParsed = page;
}
```

9.23.10 initParsePatch

⟨*hypertex*⟩+≡

```
void initParsePatch(HyperDocPage *page) {
    gEndedPage = gInDesc = gStringValueOk = gInIf =
        gInButton = gInOptional = gInVerbatim = gInPaste = gInItems =
        gInSpadsrc = FALSE;
    gParserMode = AllMode;
    gParserRegion = Scrolling;
    initTopGroup();
    clearBeStack();
    cur_spadcom = NULL;
    gLinkHashTable = page->fLinkHashTable;
    gPageBeingParsed = page;
}
```

9.23.11 parsePage

⟨*hypertex*⟩+≡

```
static void parsePage(HyperDocPage *page) {
    initParsePage(page);
    /* Get the name of the page */
    getExpectedToken(Page);
    getExpectedToken(Lbrace);
    getExpectedToken(Word);
    if (page->name == NULL)
        page->name = allocString(token.id);
    getExpectedToken(Rbrace);
    /* parse the title */
    gWindow->fDisplayedWindow = gWindow->fMainWindow;
    parseTitle(page);
    /*
     * Now start parsing the header region
     */
    parseHeader(page);
}

/*
 */
```

9.23.12 parseHyperDoc

The general HyperDoc parsing function. expects to see anything. This function will parse until it sees either:

1. A new page starting
2. An end of file
3. a closing bracket "}"

⟨*hypertex*⟩+≡

```
void parseHyperDoc(void) {
    TextNode *node = NULL /*, *save_node = NULL, *arg_node = NULL*/ ;
    for(;;) {
        ret_val = getToken();
        if (ret_val == EOF)
            return;
        switch (token.type) {
          case Spadsrc:
            parseSpadsrc(curr_node);
            break;
          case Helppage:
            parseHelp();
            break;
          case Endpatch:
          case Endpaste:
          case Rbrace:
            return;
          case Paste:
            parsePaste();
            break;
          case Pastebutton:
            parsePastebutton();
            break;
          case Endpage:
          case NewCommand:
          case Page:
            endAPage();
            return;
          case EndScroll:
            token.type = Endscroll;
          case Endscroll:
            startFooter();
            break;
          case Beginscroll:
            startScrolling();
```

```
  break;
case Thispage:        /* it really is just a word */
  curr_node->type = Word;
  curr_node->data.text = allocString(gPageBeingParsed->name);
  break;
case Icorrection:
  node->type = Noop;
  break;
case Newcond:
  parseNewcond();
  break;
case Setcond:
  parseSetcond();
  break;
case Dollar:
  parseVerbatim(Math);
  break;
case Verbatim:
  parseVerbatim(Verbatim);
  break;
case Ifcond:
  parseIfcond();
  break;
case Fi:
  if (gInIf)
      return;
  else {
      curr_node->type = Noop;
      /* Oops I had a problem parsing this puppy */
      fprintf(stderr,"(HyperDoc) \\fi found without macthing if?\n");
      longjmp(jmpbuf, 1);
      fprintf(stderr, "(HyperDoc) Longjmp failed -- Exiting \n");
      exit(-1);
  }
case Else:
  if (gInIf)
      return;
  else {
      /* Oops I had a problem parsing this puppy */
      curr_node->type = Noop;
      fprintf(stderr,
               "(HyperDoc) \\else found without macthing if?\n");
      longjmp(jmpbuf, 1);
      fprintf(stderr, "(HyperDoc) Longjmp failed -- Exiting \n");
      exit(-1);
  }
```

```
case Macro:
  parseMacro();
  break;
case Env:
  /** In this case, get the environment value, and make it a word **/
  parseEnv(curr_node);
  break;
case WindowId:
  curr_node->type = WindowId;
  curr_node->space = token.id[-1];
  curr_node->data.text = windowId(gWindow->fMainWindow);
  break;
case Punctuation:
case Word:
case Lsquarebrace:
case Dash:
  curr_node->type = token.type;
  curr_node->space = token.id[-1];
  curr_node->data.text = allocString(token.id);
  break;
case Pagename:
  {
      char *str;

      curr_node->type = Word;
      curr_node->space = 0;
      str = halloc(strlen(cur_page->name) + 1, "parse");
      sprintf(str, "%s", cur_page->name);
      curr_node->data.text = allocString(str);
      break;
  }
case Examplenumber:
  {
      char *str;
      curr_node->type = Word;
      curr_node->space = 0;
      str = halloc(5, "parse");
      sprintf(str, "%d", example_number);
      curr_node->data.text = allocString(str);
      break;
  }
case Rsquarebrace:
  if (gInOptional)
      return;
  else {
      curr_node->type = token.type;
```

```
        curr_node->space = token.id[-1];
        curr_node->data.text = allocString(token.id);
    }
    break;
  case EndTitems:
    token.type = Endtitems;
  case Endtitems:
    if (gParserMode != AllMode) {
        curr_node->type = Noop;
        fprintf(stderr,
         "(HyperDoc) Found a bad token %s\n", token_table[token.type]);
        longjmp(jmpbuf, 1);
    }
    else {
        curr_node->type = token.type;
        break;
    }
  case EndItems:
    token.type = Enditems;
  case Enditems:
    gInItems--;
  case Horizontalline:
  case Par:
  case Newline:
  case Titem:
    if (gParserMode != AllMode) {
        curr_node->type = Noop;
        fprintf(stderr,
        "(HyperDoc) Found a bad token %s\n", token_table[token.type]);
        longjmp(jmpbuf, 1);
    }
    else {
        curr_node->type = token.type;
        break;
    }
  case Begintitems:
  case Beginitems:
    if (gParserMode != AllMode) {
        curr_node->type = Noop;
        fprintf(stderr,
         "(HyperDoc) Found a bad token %s\n", token_table[token.type]);
        longjmp(jmpbuf, 1);
    }
    else {
        parseBeginItems();
        break;
```

```
    }
  case Item:
    parseItem();
    break;
  case Mitem:
    parseMitem();
    break;
  case VSpace:
  case Tab:
  case HSpace:
  case Indent:
  case Indentrel:
    parseValue1();
    break;
  case Space:
    parseValue2();
    break;
  case Lbrace:
    curr_node->type = Group;
    curr_node->space = token.id[-1];
    pushGroupStack();
    node = allocNode();
    curr_node->next = node;
    curr_node = curr_node->next;
    parseHyperDoc();
    curr_node->type = Endgroup;
    popGroupStack();
    break;
  case Upbutton:
  case Returnbutton:
  case Link:
  case Downlink:
  case Memolink:
  case Windowlink:
    parseButton();
    break;
  case Unixlink:
  case LispMemoLink:
  case LispDownLink:
  case Lisplink:
  case Lispcommand:
  case Lispcommandquit:
  case Spadlink:
  case Spaddownlink:
  case Spadmemolink:
  case Unixcommand:
```

```
case Spadcall:
case Spadcallquit:
case Qspadcall:
case Qspadcallquit:
case Lispwindowlink:
  parseCommand();
  break;
case Controlbitmap:
case Inputbitmap:
case Inputpixmap:
case Inputimage:
  parseInputPix();
  break;
case Box:
  parseBox();
  break;
case Mbox:
  parseMbox();
  break;
case Free:
  parseFree();
  break;
case Center:
  parseCenterline();
  break;
case Bound:
  addDependencies();
  break;
case Spadcommand:
case Spadgraph:
  parseSpadcommand(curr_node);
  break;
case Table:
  parseTable();
  break;
case Beep:
case Emphasize:
case BoldFace:
case Rm:
case It:
case Tt:
case Sl:
  curr_node->type = token.type;
  curr_node->space = token.id[-1];
  break;
case Inputstring:
```

```
        parseInputstring();
        break;
      case SimpleBox:
        parseSimplebox();
        break;
      case BoxValue:
      case StringValue:
        if (!gStringValueOk) {
            strcpy(ebuffer,"(HyperDoc): Unexpected Value Command:");
            strcat(ebuffer, token.id);

            parserError(ebuffer);
            curr_node->type = Noop;
            longjmp(jmpbuf, 1);
        }
        curr_node->type = token.type;
        curr_node->space = token.id[-1];
        getExpectedToken(Lbrace);
        getExpectedToken(Word);
        curr_node->data.text = allocString(token.id);
        getExpectedToken(Rbrace);
        break;
      case NoLines:
        gPageBeingParsed->pageFlags |= NOLINES;
        break;
      case Pound:
        curr_node->type = Pound;
        curr_node->space = token.id[-1];
        curr_node->next = allocNode();
        curr_node = curr_node->next;
        parseParameters();
        break;
      case Radiobox:
        parseRadiobox();
        break;
      case Radioboxes:
        parseRadioboxes();
        break;
      case Replacepage:
        parseReplacepage();
        break;
      default:
        fprintf(stderr,
            "(HyperDoc) Keyword not currently supported: %s\n", token.id);
        printPageAndFilename();
        curr_node->type = Noop;
```

```
            break;
        }
        if (gEndedPage)
            return;
        if (curr_node->type != Noop) {
            node = allocNode();
            curr_node->next = node;
            curr_node = node;
        }
    }
}
```

9.23.13 parsePageFromSocket

Parse a page from a socket source.

⟨*hypertex*⟩+≡

```
HyperDocPage *parsePageFromSocket(void) {
    HyperDocPage *page = allocPage((char *) NULL);
    HyperDocPage *hpage;
    initScanner();
    inputType = FromSpadSocket;
    inputString = "";
    cur_spadcom = NULL;
    gLinkHashTable = page->fLinkHashTable;
    hashInit(
              gLinkHashTable,
              LinkHashSize,
              (EqualFunction) windowEqual,
              (HashcodeFunction) windowCode);
    gPageBeingParsed = page;
    replace_page = NULL;
    if (setjmp(jmpbuf)) {
        /* Ooops, somewhere I had an error */
        freePage(page);
        page = (HyperDocPage *) hashFind(gWindow->fPageHashTable, "ErrorPage");
        resetConnection();
    }
    else {
        parsePage(page);
        page->type = SpadGen;
        page->filename = NULL;
        /* just for kicks, let me add this thing to the hash file */
        hpage = (HyperDocPage *) hashFind(gWindow->fPageHashTable, page->name);
        if (hpage)
            hashReplace(gWindow->fPageHashTable, (char *)page, page->name);
        else {
            hashInsert(gWindow->fPageHashTable, (char *)page, page->name);
        }
    }
    if (replace_page != NULL) {
        freePage(page);
        page = (HyperDocPage *)hashFind(gWindow->fPageHashTable, replace_page);
        if (page == NULL)
            fprintf(stderr, "(HyperDoc) Unknown page: %s\n", replace_page);
    }
    return page;
}
```

9.23.14 parsePageFromUnixfd

⟨*hypertex*⟩+≡

```
HyperDocPage *parsePageFromUnixfd(void) {
    HyperDocPage *page = allocPage((char *) NULL);
    initScanner();
    inputType = FromUnixFD;
    cur_spadcom = NULL;
    gLinkHashTable = page->fLinkHashTable;
    hashInit(
              gLinkHashTable,
              LinkHashSize,
              (EqualFunction) windowEqual,
              (HashcodeFunction) windowCode);
    gPageBeingParsed = page;
    if (setjmp(jmpbuf)) {
        /* Ooops, somewhere I had an error */
        freePage(page);
        page = (HyperDocPage *) hashFind(gWindow->fPageHashTable, "ErrorPage");
        resetConnection();
    }
    else {
        parsePage(page);
        page->type = Unixfd;
        page->filename = NULL;
    }
    return page;
}
```

9.23.15 startScrolling

⟨*hypertex*⟩+≡

```
static void startScrolling(void) {
    /*
     * if I am here than I had a begin scroll. This means I should end the
     * header, and then start parsing the footer
     */
    if (gParserRegion != Header) {
        curr_node->type = Noop;
        fprintf(stderr,
                    "(HyperDoc) Parser Error: Unexpected BeginScrollFound\n");
        longjmp(jmpbuf, 1);
        fprintf(stderr, "(HyperDoc) Longjump failed exiting\n");
    }
    curr_node->type = Endheader;
    curr_node->next = NULL;
    PopMR();
    PushMR();
    gParserRegion = Scrolling;
    gWindow->fDisplayedWindow = gWindow->fScrollWindow;
    curr_node = allocNode();
    gPageBeingParsed->scrolling = curr_node;
    curr_node->type = Scrollingnode;
}
```

9.23.16 startFooter

⟨*hypertex*⟩+≡

```
static void startFooter(void) {
    /*
     * This ends the parsing of the scrolling region, and then starts to
     * parse the footer
     */
    if (gParserRegion != Scrolling) {
        curr_node->type = Noop;
        fprintf(stderr,
                    "(HyperDoc) Parser Error: Unexpected Endscroll Found\n");
        printPageAndFilename();
        longjmp(jmpbuf, 1);
        fprintf(stderr, "(HyperDoc) Longjump failed exiting\n");
    }
    curr_node->type = Endscrolling;
    curr_node->next = NULL;
    PopMR();
    linkScrollBars();
    PushMR();
    gParserRegion = Footer;
    curr_node = allocNode();
    curr_node->type = Footernode;
    gPageBeingParsed->footer = curr_node;
    gWindow->fDisplayedWindow = gWindow->fMainWindow;
}
```

9.23.17 endAPage

⟨*hypertex*⟩+≡

```
static void endAPage(void) {
    if (gParserRegion == Scrolling) {
        fprintf(stderr, "%s\n",
                "(HyperDoc) endAPage: Unexpected End of Page occurred \
                   inside a \beginscroll");
        printPageAndFilename();
        jump();
    }
    gEndedPage = TRUE;
    if (gParserRegion == Footer) {
        /* the person had all the regions, I basically just have to leave */
        curr_node->type = Endscrolling;
        curr_node->next = NULL;
        PopMR();
    }
    else if (gParserRegion == Header) {
        /* person had a header. So just end it and return */
        curr_node->type = Endheader;
        curr_node->next = NULL;
        PopMR();
        gPageBeingParsed->scrolling = NULL;
        gPageBeingParsed->footer = NULL;
    }
}
```

9.23.18 parseReplacepage

⟨*hypertex*⟩+≡

```
static void parseReplacepage(void) {
    getExpectedToken(Lbrace);
    getToken();
    replace_page = allocString(token.id);
    getExpectedToken(Rbrace);
}
```

9.23.19 windowEqual

Hash functions for active link windows.

⟨*hypertex*⟩+≡
```
int windowEqual(Window *w1, Window *w2) {
    return *w1 == *w2;
}
```

9.23.20 windowCode

Hash code for a window.

⟨*hypertex*⟩+≡
```
int windowCode(Window *w, int size) {
    return (*w) % size;
}
```

9.23.21 windowId

⟨*hypertex*⟩+≡
```
char *windowId(Window w) {
    char *ret;
    char buff[32];
    int length;
    sprintf(buff, "%ld", w);
    length = strlen(buff);
    ret = (char *) halloc(length * sizeof(char) + 1, "windowid");
    strcpy(ret, buff);
    return (ret);
}
```

9.23.22 readHtDb

This procedure reads the ht database. It makes repeated calls to dbFileOpen, and while the returned pointer is not null, it continues to read the presented data base files.

⟨*hypertex*⟩+≡

```
void readHtDb(HashTable *page_hash, HashTable *macro_hash,
                HashTable *patch_hash) {
    FILE *db_fp;
    char dbFile[256];
    int i = 0;
    gDatabasePath = NULL;
    hashInit(
              page_hash,
              PageHashSize,
              (EqualFunction) stringEqual,
              (HashcodeFunction) stringHash);
    hashInit(
              macro_hash,
              MacroHashSize,
              (EqualFunction) stringEqual,
              (HashcodeFunction) stringHash);
    hashInit(
              patch_hash,
              PatchHashSize,
              (EqualFunction) stringEqual,
              (HashcodeFunction) stringHash);
    /* Lets initialize the FileHashTable         */
    hashInit(
              &ht_gFileHashTable,
              htfhSize,
              (EqualFunction) stringEqual,
              (HashcodeFunction) stringHash);
    while ((db_fp = dbFileOpen(dbFile)) != NULL) {
        i++;
        readHtFile(page_hash, macro_hash, patch_hash, db_fp, dbFile);
        fclose(db_fp);
    }
    if (!i) {
        fprintf(stderr,
          "(HyperDoc) readHtDb: No %s file found\n", dbFileName);
        exit(-1);
    }
    freeHash(&ht_gFileHashTable, (FreeFunction)freeString);
}
```

9.23.23 readHtFile

This procedure reads a single HyperDoc database file. It is passed an already initilaized file pointer. It reads the whole file, updating the page hash, or the macro hash only when a previous entry with the same name is not found

⟨*hypertex*⟩+≡

```
static void readHtFile(HashTable *page_hash, HashTable *macro_hash,
                        HashTable *patch_hash, FILE *db_fp, char *dbFile) {
    char filename[256];
    char *fullname = filename;
    UnloadedPage *page;
    MacroStore *macro;
    PatchStore *patch;
    int pages = 0, c, mtime, ret_val;
    struct stat fstats;
    /*fprintf(stderr,"parse-aux:readHtFile: dp_file=%s\n",dbFile);*/
    cfile = db_fp;
    initScanner();
    ret_val = strlen(dbFile) - 1;
    for (; ret_val >= 0; ret_val--)
        if (dbFile[ret_val] == '/') {
            dbFile[ret_val] = '\0';
            break;
        }
    c = getc(db_fp);
    do {
        if (c == '\t') {
            getFilename();
            fullname = allocString(token.id);
            if (fullname[0] != '/') {
                strcpy(filename, dbFile);
                strcat(filename, "/");
                strcat(filename, fullname);
                free(fullname);
                fullname = allocString(filename);
            }
            /*
             * Until I get a filename that I have not seen before, just keep
             * reading
             */
            while (hashFind(&ht_gFileHashTable, fullname) != NULL) {
                do {
                    c = getc(db_fp);
                } while ((c != EOF) && (c != '\t'));
                if (c == EOF)
```

```
        return;
    getFilename();
    fullname = allocString(token.id);
    if (fullname[0] != '/') {
        strcpy(filename, dbFile);
        strcat(filename, "/");
        strcat(filename, fullname);
        free(fullname);
        fullname = allocString(filename);
    }
}
/*fprintf(stderr,"parse-aux:readHtFile: fullname=%s\n",fullname);*/
/* If I got here, then I must have a good filename  */
hashInsert(&ht_gFileHashTable, fullname, fullname);
ret_val = stat(fullname, &fstats);
if (ret_val == -1) {
    char buffer[300];
    sprintf(buffer,
             "(HyperDoc) readHtDb: Unable To Open %s :", fullname);
    perror(buffer);
    exit(-1);
}
getToken();
mtime = atoi(token.id);
if (gverify_dates & (fstats.st_mtime > mtime)) {
    fprintf(stderr,
      "(HyperDoc) readHtFile: HyperDoc file %s has been updated\n",
      fullname);
    fprintf(stderr,
      "(HyperDoc) Issue htadd %s to update database\n", fullname);
    exit(-1);
}
while ((c = getc(db_fp)) != EOF) {
    if (c == '\t')
        break;
    ungetc(c, db_fp);
    getToken();
    switch (token.type) {
      case Page:
        getToken();

        /*
         * now check to see if the page has already been
         * loaded
         */
        page = (UnloadedPage *) halloc(sizeof(UnloadedPage),
```

```
                                    "UnloadedPage");
    page->fpos.name = allocString(fullname);
    page->name = allocString(token.id);
    getToken();
    if (hashFind(page_hash, page->name) != NULL) {
        fprintf(stderr,
            "(HyperDoc) Page name %s  occurred twice\n",
            page->name);
        fprintf(stderr,
            "(HyperDoc) The Version in %s is being ignored \n",
            page->fpos.name);
        free(page);
        getToken();
        break;
    }
    page->fpos.pos = atoi(token.id);
    getToken();
    page->fpos.ln = atoi(token.id);
    page->type = UnloadedPageType;
    hashInsert(page_hash, (char *)page, page->name);
    pages++;
    break;
  case NewCommand:
    getToken();
    macro = (MacroStore *) halloc(sizeof(MacroStore),
            "MacroStore");
    macro->fpos.name = allocString(fullname);
    macro->name = allocString(token.id);
    macro->macro_string = NULL;
    getToken();
    if (hashFind(macro_hash, macro->name) != NULL) {
        if (strcmp(macro->name, "localinfo") != 0) {
          fprintf(stderr,
              "(HyperDoc) Macro name %s  occurred twice\n",
              macro->name);
          fprintf(stderr,
            "(HyperDoc) The Version in %s is being ignored \n",
             macro->fpos.name);
        }
        getToken();
        free(macro);
        break;
    }
    macro->fpos.pos = atoi(token.id);
    getToken();
    macro->fpos.ln = atoi(token.id);
```

```
                    macro->loaded = 0;
                    hashInsert(macro_hash, (char *)macro, macro->name);
                    break;
                  case Patch:
                    getToken();
                    patch = (PatchStore *) allocPatchstore();
                    patch->fpos.name = allocString(fullname);
                    patch->name = allocString(token.id);
                    getToken();
                    patch->fpos.pos = atoi(token.id);
                    getToken();
                    patch->fpos.ln = atoi(token.id);
                    if (hashFind(patch_hash, patch->name) != NULL) {
                      fprintf(stderr,
                        "(HyperDoc) Patch name %s  occurred twice\n",
                        patch->name);
                      fprintf(stderr,
                        "(HyperDoc) The version in %s is being ignored \n",
                        patch->fpos.name);
                      freePatch(patch);
                      break;
                    }
                    hashInsert(patch_hash, (char *)patch, patch->name);
                    break;
                  default:
                    fprintf(stderr,
                      "(HyperDoc) readHtDb: Unknown type %s in ht.db\n",
                      token.id);
                    exit(-1);
                    break;
                }
            }
        }
        else
            c = getc(db_fp);
    } while (c != EOF);
/*    fprintf(stderr,
     "parse-aux:readHtFile:read %d pages from database\n", pages); */
}
```

9.23.24 makeLinkWindow

Create an unmapped input-only window for an active screen area.

⟨*hypertex*⟩+≡

```
HyperLink *makeLinkWindow(TextNode *link_node, int type, int isSubWin) {
    HyperLink *link;
    XSetWindowAttributes at;
    if (make_input_file)
        switch (type) {
          case Downlink:
          case Memolink:
          case Windowlink:{
                char *name;
                HyperDocPage *p;

                name = printToString(link_node);
                p = (HyperDocPage *) hashFind(gWindow->fPageHashTable, name);
                if (!p)
                    printf("undefined link to %s\n", name);
                break;
            }
        }
    else {
        link = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink");
        if (link == NULL) {
            fprintf(stderr,
              "(HyperDoc) Ran out of memory allocating a hypertext link!\n");
            exit(-1);
        }
        at.cursor = gActiveCursor;
        at.event_mask = ButtonPress;
        if (isSubWin)
          link->win =
            XCreateWindow(gXDisplay, gWindow->fDisplayedWindow, 0, 0,
                          100, 100, 0, 0, InputOnly, CopyFromParent,
                          CWEventMask | CWCursor, &at);
        else
           link->win = 0;
        link->type = type;
        link->x = link->y = 0;
        link->reference.node = link_node;
        hashInsert(gLinkHashTable, (char *)link,(char *)&link->win);
        return link;
    }
    return 0;
```

```
}
```

9.23.25 makePasteWindow

⟨*hypertex*⟩+≡

```
HyperLink *makePasteWindow(PasteNode *paste) {
    HyperLink *link;
    XSetWindowAttributes at;
    if (!make_input_file) {
        link = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink");
        if (link == NULL) {
            fprintf(stderr,
               "(HyperDoc) Ran out of memory allocating a hypertext link!\n");
            exit(-1);
        }
        at.cursor = gActiveCursor;
        at.event_mask = ButtonPress;
        link->win = XCreateWindow(gXDisplay, gWindow->fDisplayedWindow,
                                  0, 0, 100, 100, 0,
                                  0, InputOnly, CopyFromParent,
                                  CWEventMask | CWCursor, &at);
        link->type = Pastebutton;
        link->x = link->y = 0;
        link->reference.paste = paste;
        hashInsert(gLinkHashTable, (char *)link,(char *) &link->win);
        return link;
    }
    return 0;
}
```

9.23.26 makeSpecialPage

Create a HyperDoc page structure with the given type and name.

⟨*hypertex*⟩+≡

```
static HyperDocPage *makeSpecialPage(int type, char *name) {
    HyperDocPage *page = allocPage(name);
    if (page == NULL) {
        fprintf(stderr, "(HyperDoc) Ran out of memory allocating page.\n");
        exit(-1);
    }
    page->type = type;
    free(page->fLinkHashTable);
    page->fLinkHashTable = NULL;
    return page;
}
```

9.23.27 main

Insert the special button page types into the page hash table.

⟨*hypertex*⟩+≡

```
void makeSpecialPages(HashTable *pageHashTable) {
    hashInsert(pageHashTable,
               (char *)makeSpecialPage(Quitbutton, "QuitPage"),
               "QuitPage");
    hashInsert(pageHashTable,
               (char *)makeSpecialPage(Returnbutton, "ReturnPage"),
               "ReturnPage");
    hashInsert(pageHashTable,
               (char *)makeSpecialPage(Upbutton, "UpPage"),
               "UpPage");
}
```

9.23.28 addDependencies

Here is where I put the item into the pages linked list. Parse the `\bound{varlist}` command, and add vars to dependency table.

⟨*hypertex*⟩+≡

```
void addDependencies(void) {
    TextNode *bound_node = curr_node;
    TextNode *node;
    SpadcomDepend *depend;
    if (cur_spadcom == NULL) {
        fprintf(stderr, "(HyperDoc) \\bound occuring outside a \\spadcom\n");
        printPageAndFilename();
        exit(-1);
    }
    curr_node->type = Bound;
    curr_node->data.node = allocNode();
    curr_node = curr_node->data.node;
    getExpectedToken(Lbrace);
    parseHyperDoc();
    curr_node->type = Endarg;
    curr_node = bound_node;
    if (gPageBeingParsed->depend_hash == NULL) {
        gPageBeingParsed->depend_hash =
            (HashTable *) halloc(sizeof(HashTable), "Hash Table");
        hashInit(
                  gPageBeingParsed->depend_hash,
                  DependHashSize,
                  (EqualFunction) stringEqual,
                  (HashcodeFunction) stringHash);
    }
    for (node = bound_node->data.node;
         node->type != Endarg;
         node = node->next) {
       if (node->type == Word) {
            depend =
              (SpadcomDepend *) halloc(sizeof(SpadcomDepend), "SpadcomDepend");
            depend->label = allocString(node->data.text);
            depend->spadcom = cur_spadcom;
            depend->executed = 0;
            hashInsert(gPageBeingParsed->depend_hash, (char *)depend,
                       depend->label);
        }
    }
}
```

9.23.29 isNumber

Returns true iff the TextNode contains a single integer.

⟨*hypertex*⟩+≡

```
int isNumber(char * str) {
    char *s;
    for (s = str; *s != '\0'; s++) {
        if (!(isdigit(*s) || *s == '-'))
            return 0;
    }
    return 1;
}
```

9.23.30 parserError

This procedure is called by the parser when an error occurs. It prints the error message, followed by the next 10 tokens to ease finding the error for the user.

⟨*hypertex*⟩+≡

```
void parserError(char *str) {
    int i, v;
    fprintf(stderr, " %s\n", str);
    fprintf(stderr, "Here are the next 10 tokens:\n");
    for (i = 0; i < 10; i++) {
        v = getToken();
        if (v == EOF)
            break;
        printToken();
    }
    fprintf(stderr, "\n");
    exit(-1);
}
```

9.23.31 getFilename

Advance token to the next token in the input stream.

⟨*hypertex*⟩+≡

```
int getFilename(void) {
    int c, ws;
    static int seen_white = 0; /*UNUSED */
    static char buffer[256];
    char *buf = buffer;
    if (last_token) {
        last_token = 0;
        return 0;
    }
    do {
        keyword_fpos = fpos;
        c = getChar();
        ws = whitespace(c);
        if (ws)
            seen_white = 1;
    } while (ws);
    switch (c) {
      case EOF:
        fprintf(stderr,
          "(HyperDoc) Error trying to read %s, unexpected end-of-file.\n",
          dbFileName);
        exit(-1);
      case '%':
      case '\\':
      case '{':
      case '}':
        fprintf(stderr, "(HyperDoc) Error unexpected character %c.\n",c);
        exit(-1);
      default:
        do {
            *buf++ = c;
        } while ((c = getChar()) != EOF && !filedelim(c));
        ungetChar(c);
        *buf = '\0';
        token.type = Word;
        token.id = buffer;
        seen_white = 0;
        break;
    }
    return 1;
}
```

9.23.32 getInputString

⟨*hypertex*⟩+≡

```
char *getInputString(void) {
    char *string;
    TextNode *string_node,*save_node;
    save_node = curr_node;
    /* Get the nodes that make up the string */
    string_node = allocNode();
    curr_node = string_node;
    parseHyperDoc();
    curr_node->type = Endarg;
    /* Once here we print to string to get the actual name */
    string = printToString(string_node);
    freeNode(string_node, 0);
    curr_node=save_node;
    return string;
}
```

9.23.33 getWhere

Tries to determine if there is an optional argument for where I should be parsing from. If so it then tries to determine which.

⟨*hypertex*⟩+≡

```
int getWhere(void) {
    int tw;
    getToken();
    if (token.type != Word)
        return -1;
    /* Now try to determine if it is a good type */
    if (!strcmp(token.id, "lisp")) {
        tw = FromSpadSocket;
    }
    else if (!strcmp(token.id, "unix")) {
        tw = FromUnixFD;
    }
    else if (!strcmp(token.id, "ht")) {
        tw = FromFile;
    }
    else {
        return -1;
    }
    /* now check to see if I got a closing square brace */
    getToken();
    if (token.type != Rsquarebrace)
        return -1;
    return tw;
}
```

9.23.34 findFp

⟨*hypertex*⟩+≡

```
FILE *findFp(FilePosition fp) {
    FILE *lfile;
    char fullname[256], addname[256];
    int ret_val;
    /* find the source file in the file hash table, if not there, open it */
    lfile = (FILE *) hashFind(&gFileHashTable, fp.name);
    if (lfile == NULL) {
        lfile = htFileOpen(fullname, addname, fp.name);
        hashInsert(&gFileHashTable, (char *)lfile, fp.name);
    }
    /* seek to beginning fp.pos */
    ret_val = fseek(lfile, fp.pos, 0);
    if (ret_val == -1) {
        perror("fseeking to a page");
        longjmp(jmpbuf, 1);
    }
    /* now set some global values */
    page_start_fpos = fp.pos;
    line_number = fp.ln;
    return lfile;
}
```

9.24 Handle InputString, SimpleBox, RadioBox input

9.24.1 makeInputWindow

⟨*hypertex*⟩+≡

```
HyperLink *makeInputWindow(InputItem * item) {
  HyperLink *link;
  XSetWindowAttributes at;
  if (!make_input_file) {
    link = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink");
    if (link == NULL) {
      fprintf(stderr, "Ran out of memory allocating a hyper link!\n");
      exit(-1);
    }
    at.cursor = gActiveCursor;
    at.background_pixel = gInputBackgroundColor;
    at.border_pixel = gActiveColor;
    link->win =
      XCreateWindow(gXDisplay, gWindow->fDisplayedWindow, 0, 0, 100, 100, 0,
                    0, InputOutput, CopyFromParent,
                    CWCursor | CWBackPixel | CWBorderPixel, &at);
    XSelectInput(gXDisplay, link->win, ButtonPressMask);
    link->type = Inputstring;
    link->x = link->y = 0;
    /** This way when I click in an input window, I need only use reference
      to get a pointer to the item                              ***/
    link->reference.string = item;
    hashInsert(gLinkHashTable,(char *) link,(char *) &link->win);
    return link;
  }
  return 0;
}

/* create an unmapped input window for boxes */
```

9.24.2 makeBoxWindow

⟨*hypertex*⟩+≡

```
HyperLink *makeBoxWindow(InputBox * box, int type) {
  HyperLink *link = 0;
  XSetWindowAttributes at;
  if (!make_input_file) {
    link = (HyperLink *) halloc(sizeof(HyperLink), "makeBoxWindow");
    if (link == NULL) {
      fprintf(stderr, "Ran out of memory allocating a hyper link!\n");
      exit(-1);
    }
    at.cursor = gActiveCursor;
    at.background_pixel = gInputBackgroundColor;
    link->win = XCreateWindow(gXDisplay, gWindow->fDisplayedWindow,
                              0, 0, 100, 100, 0,
                              0, InputOutput, CopyFromParent,
                              CWCursor | CWBackPixel, &at);
    XSelectInput(gXDisplay, link->win, ButtonPressMask);
    link->type = type;
    link->x = link->y = 0;
    /** This way when I click in an input window, I need only use reference
      to get a pointer to the item                              ***/
    link->reference.box = box;
    hashInsert(gLinkHashTable, (char *)link,(char *) &link->win);
  }
  return link;
}
```

9.24.3 initializeDefault

⟨*hypertex*⟩+≡

```
void initializeDefault(InputItem *item,char * buff) {
  LineStruct *newline;
  LineStruct *curr_line;
  int size = item->size;
  int bp;
  item->curr_line = item->lines = allocInputline(size);
  curr_line = item->lines;
  item->num_lines = 1;
  curr_line->line_number = 1;
  /* while I still have lines to fill */
  for (bp = 0; *buff;) {
    if (*buff == '\n') {
      curr_line->len = bp;
      curr_line->buffer[bp] = 0;
      newline = allocInputline(size);
      newline->line_number = ++(item->num_lines);
      curr_line->next = newline;
      newline->prev = curr_line;
      curr_line = newline;
      bp = 0;
      buff++;
    }
    else if (bp == size) {
      curr_line->len = size + 1;
      curr_line->buffer[size] = '_';
      curr_line->buffer[size + 1] = 0;
      newline = allocInputline(size);
      newline->line_number = ++(item->num_lines);
      curr_line->next = newline;
      newline->prev = curr_line;
      bp = 0;
      curr_line = newline;
    }
    else {
      curr_line->buffer[bp++] = *buff++;
    }
  }
  curr_line->buff_pntr = curr_line->len = bp;
  item->curr_line = curr_line;
}
```

9.24.4 parseInputstring

Parse the input string statement.

⟨*hypertex*⟩+≡

```
void parseInputstring(void) {
  TextNode *input_node = curr_node;
  char *name;
  InputItem *item;
  int size;
  char *default_value;
  gStringValueOk = 0;
  /* first get the name */
  input_node->type = token.type;
  getExpectedToken(Lbrace);
  name = getInputString();
  input_node->data.text = allocString(name);
  /* now get the width */
  getExpectedToken(Lbrace);
  getExpectedToken(Word);
  getExpectedToken(Rbrace);
  size = atoi(token.id);
  if (size < 0) {
    fprintf(stderr, "Illegal size in Input string\n");
    longjmp(jmpbuf, 1);
  }
  /* get the default value */
  getExpectedToken(Lbrace);
  default_value = getInputString();
  /** now I need to malloc space for the input stuff **/
  item = (InputItem *) halloc(sizeof(InputItem), "InputItem");
  /* Now store all the string info */
  item->name = (char *)
    halloc((strlen(input_node->data.text) + 1) * (sizeof(char)),
           "parseInputstring");
  strcpy(item->name, input_node->data.text);
  item->size = size;
  item->entered = 0;
  item->next = NULL;
  initializeDefault(item, default_value);
  /** Now that I have all the structures made, lets make the window, and
    add the item to the list                                    ****/
  input_node->link = makeInputWindow(item);
  if (!make_input_file)
    item->win = input_node->link->win;      /* TTT */
  insertItem(item);
```

```
    gStringValueOk = 1;
    curr_node = input_node;
    return ;
  }
```

9.24.5 parseSimplebox

⟨*hypertex*⟩+≡

```
void parseSimplebox(void) {
  InputBox *box;
  char *name;
  short int picked = 0;
  char *filename;
  TextNode *input_box = curr_node;
  gStringValueOk = 0;
  /* set the type and space fields  */
  input_box->type = SimpleBox;
  input_box->space = token.id[-1];
  /* IS it selected? */
  getToken();
  if (token.type == Lsquarebrace) {
    getExpectedToken(Word);
    if (!isNumber(token.id)) {
      fprintf(stderr,"parse_simple_box: Expected a value not %s\n", token.id);
      printPageAndFilename();
      jump();
    }
    else if (!strcmp(token.id, "1"))
      picked = 1;
    else if (!strcmp(token.id, "0"))
      picked = 0;
    else {
      fprintf(stderr, "parse_simple_box: Unexpected Value %s\n", token.id);
      printPageAndFilename();
      jump();
    }
    getExpectedToken(Rsquarebrace);
    getToken();
  }
  if (token.type != Lbrace) {
    tokenName(token.type);
    fprintf(stderr, "parse_inputbox was expecting a { not a %s\n", ebuffer);
    printPageAndFilename();
    jump();
  }
  name = getInputString();
  if (gPageBeingParsed->box_hash && hashFind(gPageBeingParsed->box_hash, name)) {
    fprintf(stderr, "Input box name %s is not unique \n", name);
    printPageAndFilename();
    jump();
  }
```

```
  box = allocInputbox();
  box->name = allocString(name);
  input_box->data.text = allocString(name);
  box->picked = picked;
  /* Get the filename for the selected and unselected bitmaps */
  getExpectedToken(Lbrace);
  filename = getInputString();
  if (!make_input_file)
    box->selected = insertImageStruct(filename);
  getExpectedToken(Lbrace);
  filename = getInputString();
  if (!make_input_file) {
    box->unselected = insertImageStruct(filename);
    /* set the width and height for the maximaum of the two */
    input_box->height = max(box->selected->height, box->unselected->height);
    input_box->width = max(box->selected->width, box->unselected->width);
    /* Make the window and stuff */
    input_box->link = makeBoxWindow(box, SimpleBox);
    box->win = input_box->link->win;
    /* Now add the box to the box_has table for this window */
    if (gPageBeingParsed->box_hash == NULL) {
      gPageBeingParsed->box_hash = (HashTable *) halloc(sizeof(HashTable),
                                                                  "Box Hash");
      hashInit(
                gPageBeingParsed->box_hash,
                BoxHashSize,
                (EqualFunction) stringEqual,
                (HashcodeFunction) stringHash);
    }
    hashInsert(gPageBeingParsed->box_hash, (char *)box, box->name);
  }
  /* reset the curr_node and then return */
  curr_node = input_box;
  gStringValueOk = 1;
  return;
}
```

9.24.6 parseRadiobox

⟨*hypertex*⟩+≡

```
void parseRadiobox(void) {
  InputBox *box;
  char *name;
  char *group_name;
  short int picked = 0;
  TextNode *input_box = curr_node;
  gStringValueOk = 0;
  /* set the type and space fields  */
  input_box->type = Radiobox;
  input_box->space = token.id[-1];
  /* IS it selected? */
  getToken();
  if (token.type == Lsquarebrace) {
    getExpectedToken(Word);
    if (!isNumber(token.id)) {
      fprintf(stderr,"parse_simple_box: Expected a value not %s\n", token.id);
      printPageAndFilename();
      jump();
    }
    else if (!strcmp(token.id, "1"))
      picked = 1;
    else if (!strcmp(token.id, "0"))
      picked = 0;
    else {
      fprintf(stderr, "parse_simple_box: Unexpected Value %s\n", token.id);
      printPageAndFilename();
      jump();
    }
    getExpectedToken(Rsquarebrace);
    getToken();
  }
  if (token.type != Lbrace) {
    tokenName(token.type);
    fprintf(stderr, "parse_inputbox was expecting a { not a %s\n", ebuffer);
    printPageAndFilename();
    jump();
  }
  name = getInputString();
  if (gPageBeingParsed->box_hash && hashFind(gPageBeingParsed->box_hash, name)) {
    fprintf(stderr, "Input box name %s is not unique \n", name);
    printPageAndFilename();
    jump();
  }
```

```
  box = allocInputbox();
  box->name = allocString(name);
  input_box->data.text = allocString(name);
  box->picked = picked;
  /* Now what I need to do is get the group name */
  getToken();
  if (token.type != Lbrace) {
    tokenName(token.type);
    fprintf(stderr, "parse_inputbox was expecting a { not a %s\n", ebuffer);
    printPageAndFilename();
    jump();
  }
  group_name = getInputString();
  /*
   * Now call a routine which searches the radio box list for the current
   * group name, and if found adds this box to it
   */
  addBoxToRbList(group_name, box);
  input_box->width = box->rbs->width;
  input_box->height = box->rbs->height;
  /* Make the window and stuff */
  input_box->link = makeBoxWindow(box, Radiobox);
  if (!make_input_file)
    box->win = input_box->link->win;        /* TTT */
  /* Now add the box to the box_has table for this window */
  if (gPageBeingParsed->box_hash == NULL) {
    gPageBeingParsed->box_hash = (HashTable *) halloc(sizeof(HashTable),
                                                                  "Box Hash");
    hashInit(
              gPageBeingParsed->box_hash,
              BoxHashSize,
              (EqualFunction) stringEqual,
              (HashcodeFunction) stringHash);
  }
  hashInsert(gPageBeingParsed->box_hash, (char *)box, box->name);
  /* reset the curr_node and then return */
  curr_node = input_box;
  gStringValueOk = 1;
  return;
}
```

9.24.7 addBoxToRbList

⟨*hypertex*⟩+≡

```
static void addBoxToRbList(char *name,InputBox *box) {
  RadioBoxes *trace = gPageBeingParsed->radio_boxes;
  InputBox *list;
  /*int found = 0;*/
  while (trace != NULL && strcmp(trace->name, name))
    trace = trace->next;
  if (!trace) {
    fprintf(stderr, "Tried to add a radio box to a non-existent group %s\n",
            name);
    printPageAndFilename();
    jump();
  }
  /* now add the box to the list */
  list = trace->boxes;
  box->next = list;
  trace->boxes = box;
  if (box->picked && checkOthers(box->next)) {
    fprintf(stderr, "Only a single radio button can be picked\n");
    printPageAndFilename();
    box->picked = 0;
  }
  box->selected = trace->selected;
  box->unselected = trace->unselected;
  box->rbs = trace;
  return;
}
```

9.24.8 checkOthers

⟨*hypertex*⟩+≡

```
static int checkOthers(InputBox *list) {
  InputBox *trace = list;
  while (trace != NULL && !trace->picked)
    trace = trace->next;
  if (trace != NULL)
    return 1;
  else
    return 0;
}
```

9.24.9 insertItem

Inserts an item into the current input list.

⟨*hypertex*⟩+≡

```
static void insertItem(InputItem *item) {
  InputItem *trace = gPageBeingParsed->input_list;
  if (gPageBeingParsed->currentItem == NULL) {
    gPageBeingParsed->currentItem = item;
  }
  if (trace == NULL) {
    /** Insert at the front of the list **/
    gPageBeingParsed->input_list = item;
    return;
  }
  else {
    /** find the end of the list **/
    while (trace->next != NULL)
      trace = trace->next;
    trace->next = item;
    return;
  }
}
```

9.24.10 initPasteItem

hypertex!initPasteItem

⟨*hypertex*⟩+≡

```
void initPasteItem(InputItem *item) {
  InputItem *trace = gPageBeingParsed->input_list;
  if (!item) {
    gPageBeingParsed->input_list = NULL;
    gPageBeingParsed->currentItem = NULL;
    save_item = NULL;
  }
  else {
    save_item = item->next;
    trace->next = NULL;
  }
}
```

9.24.11 repasteItem

⟨*hypertex*⟩+≡

```
void repasteItem(void) {
  InputItem *trace;
  if (save_item) {
    for (trace = gPageBeingParsed->input_list; trace && trace->next != NULL;
         trace = trace->next);
    if (trace) {
      trace->next = save_item;
    }
    else {
      gWindow->page->input_list = save_item;
      gWindow->page->currentItem = save_item;
    }
  }
  save_item = NULL;
}
```

9.24.12 currentItem

⟨*hypertex*⟩+≡

```
InputItem *currentItem(void) {
  InputItem *trace = gPageBeingParsed->input_list;
  if (trace) {
    for (; trace->next != NULL; trace = trace->next);
    return trace;
  }
  else
    return NULL;
}
```

9.24.13 alreadyThere

⟨*hypertex*⟩+≡

```
int alreadyThere(char *name) {
  RadioBoxes *trace = gPageBeingParsed->radio_boxes;
  while (trace && strcmp(trace->name, name))
    trace = trace->next;
  if (trace)
    return 1;
  else
    return 0;
}
```

9.24.14 parseRadioboxes

⟨*hypertex*⟩+≡

```
void parseRadioboxes(void) {
  TextNode *return_node = curr_node;
  RadioBoxes *newrb;
  char *fname;
  /* I really don't need this node, it just sets up some parsing stuff */
  return_node->type = Noop;
  newrb = allocRbs();
  getToken();
  if (token.type != Lbrace) {
    tokenName(token.type);
    fprintf(stderr, "\\radioboxes was expecting a name not %s\n", ebuffer);
    printPageAndFilename();
    jump();
  }
  newrb->name = allocString(getInputString());
  /* quick search for the name in the current list */
  if (alreadyThere(newrb->name)) {
    free(newrb->name);
    free(newrb);
    fprintf(stderr, "Tried to redefine radioboxes %s\n", newrb->name);
    printPageAndFilename();
    jump();
  }
  /* now I have to get the selected and unslected bitmaps */
  getToken();
  if (token.type != Lbrace) {
    tokenName(token.type);
    fprintf(stderr, "\\radioboxes was expecting a name not %s\n", ebuffer);
    printPageAndFilename();
    jump();
  }
  fname = getInputString();
  if (!make_input_file)
    newrb->selected = insertImageStruct(fname);
  getToken();
  if (token.type != Lbrace) {
    tokenName(token.type);
    fprintf(stderr, "\\radioboxes was expecting a name not %s\n", ebuffer);
    printPageAndFilename();
    jump();
  }
  fname = getInputString();
  if (!make_input_file) {
```

```
      newrb->unselected = insertImageStruct(fname);
      newrb->height = max(newrb->selected->height, newrb->unselected->height);
      newrb->width = max(newrb->selected->width, newrb->unselected->width);
      /* now add the thing to the current list of radio boxes */
    }
    newrb->next = gPageBeingParsed->radio_boxes;
    gPageBeingParsed->radio_boxes = newrb;
    curr_node = return_node;
    return;
  }
```

9.25 Routines for paste-in areas

9.25.1 parsePaste

⟨*hypertex*⟩+≡

```
void parsePaste(void) {
    TextNode *pn = curr_node;
    PasteNode *paste;
    int where;
    if (gParserRegion != Scrolling) {
        fprintf(stderr,
          "(HyperDoc) Paste areas are only allowed in the scrolling area:");
        printPageAndFilename();
        jump();
    }
    gInPaste++;
    /* now I need to get the name */
    getToken();
    if (token.type != Lbrace) {
        fprintf(stderr, "(HyperDoc) A paste area needs a name:\n");
        printNextTenTokens();
        printPageAndFilename();
        jump();
    }
    pn->data.text = allocString(getInputString());
    pn->type = Paste;
    /*
     * now see if there is already an entry in the hash_table for this thing,
     * if not create it and put it there.
     */
    paste = (PasteNode *) hashFind(gWindow->fPasteHashTable, pn->data.text);
    if (paste == 0) {
        paste = allocPasteNode(pn->data.text);
        hashInsert(gWindow->fPasteHashTable, (char *)paste, paste->name);
    }
    else if (paste->haspaste) {
      fprintf(stderr,
                "(HyperDoc) Tried to redefine paste area %s\n", paste->name);
      printPageAndFilename();
      /* jump(); */
    }
    paste->haspaste = 1;
    paste->paste_item = currentItem();
    getToken();
    if (token.type == Lsquarebrace) {
        /* user wishes to specify a where to send the command */
```

```
        where = getWhere();
        if (where == -1) {
            paste->where = -1;
            fprintf(stderr,
                 "(HyperDoc) \\begin{paste} was expecting [lisp|unix|ht]\n");
            printNextTenTokens();
            printPageAndFilename();
            jump();
        }
        else
            paste->where = where;
        getToken();
    }
    else
        paste->where = FromFile;
    /* now try to get the command argument or page name */
    if (token.type != Lbrace) {
        paste->where = 0;
        fprintf(stderr,
                    "(HyperDoc) \\begin{paste} was expecting an argument\n");
        printNextTenTokens();
        printPageAndFilename();
        jump();
    }
    paste->arg_node = allocNode();
    curr_node = paste->arg_node;
    parseHyperDoc();
    curr_node->type = Endarg;
    gWindow->fDisplayedWindow = gWindow->fScrollWindow;
    /* Now try to find the displaying text */
    pn->next = allocNode();
    curr_node = pn->next;
    parseHyperDoc();
    curr_node->type = Endpaste;
    paste->end_node = curr_node;
    paste->begin_node = pn;
    gInPaste--;
}
```

9.25.2 parsePastebutton

⟨*hypertex*⟩+≡

```
void parsePastebutton(void) {
    PasteNode *paste;
    TextNode *pb;
    /*
     * this routine parse a \pastebutton expression. The syntax is
     * \pastebutton{name}
     */
    pb = curr_node;
    pb->type = Pastebutton;
    /* first thing I should do is get the name */
    getToken();
    if (token.type != Lbrace) {
        fprintf(stderr, "(HyperDoc) \\pastebutton needs a name\n");
        printPageAndFilename();
        printNextTenTokens();
        jump();
    }
    pb->data.text = allocString(getInputString());
    /*
     * now I should see if the paste area has already been parsed, and if not
     * I should create a spot in the hash table for it
     */
    paste = (PasteNode *) hashFind(gWindow->fPasteHashTable, pb->data.text);
    if (paste == 0) {
        paste = allocPasteNode(pb->data.text);
        hashInsert(gWindow->fPasteHashTable,(char *) paste, paste->name);
    }
    else if (paste->hasbutton) {
      fprintf(stderr,
               "(HyperDoc) Tried to redefine paste area %s\n", paste->name);
      printPageAndFilename();
      /* jump(); */
    }
    paste->hasbutton = 1;
    /* Now we need to parse the HyperDoc and for the displayed text */
    getToken();
    if (token.type != Lbrace) {
        fprintf(stderr, "(HyperDoc) \\pastebutton was expecting a { \n");
        printPageAndFilename();
        printNextTenTokens();
        jump();
    }
    pb->next = allocNode();
```

```
    curr_node = pb->next;
    parseHyperDoc();
    curr_node->type = Endpastebutton;
    /* once that is done I need only make the window for this link */
    pb->link = makePasteWindow(paste);
}
```

9.25.3 parsePatch

This routine is responsible for parsing a patch from a file. To do this I guess er will initScanner, then parse, the parsed piece of text will replace the current PasteNode which will be squashed down to nothing, and then discarded.

⟨*hypertex*⟩+≡

```
HyperDocPage *parsePatch(PasteNode *paste) {
    TextNode *new;
    TextNode *end_node;
    TextNode *begin_node;
    TextNode *arg_node;
    TextNode *throw;
    TextNode *next_node;
    InputItem *paste_item = paste->paste_item;
    int where = paste->where;
    GroupItem *g = paste->group;
    ItemStack *is = paste->item_stack;
    PatchStore *patch;
    char *patch_name;
    int ret_value = 1;
    /* prepare to throw away the current paste node */
    end_node = paste->end_node;
    next_node = end_node->next;
    begin_node = paste->begin_node;
    arg_node = paste->arg_node;
    throw = begin_node->next;
    /* now read the new stuff and add it in between all this stuff */
    switch (where) {
      case FromFile:
        patch_name = printToString(arg_node);
        patch = (PatchStore *) hashFind(gWindow->fPatchHashTable, patch_name);
        if (!patch) {
            fprintf(stderr, "(HyperDoc) Unknown patch name %s\n", patch_name);
            BeepAtTheUser();
            return 0;
        }
        if (!patch->loaded)
            loadPatch(patch);
        inputType = FromString;
        inputString = patch->string;
        break;
      case FromSpadSocket:
        inputType = FromSpadSocket;
        ret_value = issueServerpaste(arg_node);
        if (ret_value < 0) {
```

```
            paste->where = where;
            paste->end_node = end_node;
            paste->arg_node = arg_node;
            paste->group = g;
            paste->item_stack = is;
            paste->haspaste = 1;
            return 0;
        }
        break;
      case FromUnixFD:
        inputType = FromUnixFD;
        issueUnixpaste(arg_node);
        break;
      default:
        fprintf(stderr, "(HyperDoc) \\parsebutton error: Unknown where\n");
        exit(-1);
        break;
    }
    paste->where = 0;
    paste->end_node = paste->arg_node = paste->begin_node = 0;
    paste->group = 0;
    paste->item_stack = 0;
    paste->haspaste = 0;
    paste->paste_item = 0;
    /* set the jump buffer in case it is needed */
    if (setjmp(jmpbuf)) {
        /*** OOOPS, an error occurred ****/
        fprintf(stderr, "(HyperDoc) Had an error parsing a patch: Goodbye!\n");
        exit(-1);
    }
    end_node->next = 0;
    freeNode(throw, 1);
    initParsePatch(gWindow->page);
    initPasteItem(paste_item);
    getToken();
    if (token.type != Patch) {
        fprintf(stderr, "(HyperDoc) Pastebutton %s was expecting a patch\n",
                paste->name);
        jump();
    }
    if (inputType == FromString) {
        getToken();
        if (token.type != Lbrace) {
            tokenName(token.type);
            fprintf(stderr, "(HyperDoc) Unexpected %s \n", ebuffer);
            printPageAndFilename();
```

```
            jump();
        }
        getToken();
        if (token.type != Word) {
            tokenName(token.type);
            fprintf(stderr, "(HyperDoc) Unexpected %s \n", ebuffer);
            printPageAndFilename();
            jump();
        }
        getToken();
        if (token.type != Rbrace) {
            tokenName(token.type);
            fprintf(stderr, "(HyperDoc) Unexpected %s \n", ebuffer);
            printPageAndFilename();
            jump();
        }
    }
    new = allocNode();
    curr_node = new;
    parseHyperDoc();
    /* Once I am back, I need only reallign all the text structures */
    curr_node->type = Noop;
    curr_node->next = next_node;
    begin_node->next = new;
    begin_node->type = Noop;
    free(begin_node->data.text);
    begin_node->data.text = 0;
    gWindow->fDisplayedWindow = gWindow->fScrollWindow;
    repasteItem();
    pastePage(begin_node);
    /* so now I should just be able to disappear */
    return gWindow->page;
}
```

9.25.4 loadPatch

⟨*hypertex*⟩+≡

```
static void loadPatch(PatchStore *patch) {
    long start_fpos;
    int size = 0;
    int limsize;
    char *trace;
    saveScannerState();
    cfile = findFp(patch->fpos);
    initScanner();
    /** First thing I should do is make sure that the name is correct ***/
    start_fpos = fpos;
    getExpectedToken(Patch);
    getExpectedToken(Lbrace);
    getExpectedToken(Word);
    if (strcmp(token.id, patch->name)) {
        /** WOW, Somehow I had the location of the wrong macro **/
        fprintf(stderr,
           "(HyperDoc) Expected patch name %s: got instead %s in loadPatch\n",
           patch->name, token.id);
        jump();
    }
    getExpectedToken(Rbrace);
    scanHyperDoc();
    fseek(cfile, patch->fpos.pos + start_fpos, 0);
    limsize = fpos - start_fpos + 1;
    patch->string =
       (char *) halloc((limsize + 1) * sizeof(char), "Patch String");
    for (size = 1, trace = patch->string; size < limsize; size++)
        *trace++ = getc(cfile);
    *trace = '\0';
    patch->loaded = 1;
    restoreScannerState();
}
```

9.26 parsing routines for node types

9.26.1 parseIfcond

⟨*hypertex*⟩+≡

```
void parseIfcond(void) {
    TextNode *ifnode = curr_node;
    TextNode *endif;
    TextNode *condnode;
    /*
     * parse a conditional. At first I am just going to parse if
     * <hypertext> fi
     */
    if (gInIf) {
        curr_node->type = Noop;
        fprintf(stderr, "\\if found within \\if \n");
        longjmp(jmpbuf, 1);
        fprintf(stderr, "Longjump failed, Exiting\n");
        exit(-1);
    }
    gInIf++;
    curr_node->type = Ifcond;
    curr_node->space = token.id[-1];
    curr_node->data.ifnode = allocIfnode();
    /* Now get the cond node I hope */
    condnode = curr_node->data.ifnode->cond = allocNode();
    curr_node = condnode;
    parseCondnode();
    endif = allocNode();
    endif->type = Endif;
    ifnode->data.ifnode->thennode = allocNode();
    curr_node = ifnode->data.ifnode->thennode;
    parseHyperDoc();
    if (token.type == Fi) {
        curr_node->type = Fi;
        curr_node->next = endif;
        ifnode->data.ifnode->elsenode = endif;
    }
    else if (token.type == Else) {
        /* first finish up the then part */
        curr_node->type = Fi;
        curr_node->next = endif;
        /* the go and parse the else part */
        ifnode->data.ifnode->elsenode = allocNode();
        curr_node = ifnode->data.ifnode->elsenode;
        parseHyperDoc();
```

```
        if (token.type != Fi) {
            tokenName(token.type);
            curr_node->type = Noop;
            fprintf(stderr, "Expected a \\fi not a %s", ebuffer);
            longjmp(jmpbuf, 1);
            fprintf(stderr, "Longjump failed, Exiting\n");
            exit(-1);
        }
        curr_node->type = Fi;
        curr_node->next = endif;
    }
    else {
        curr_node->type = Noop;
        tokenName(token.type);
        fprintf(stderr, "Expected a \\fi not a %s", ebuffer);
        longjmp(jmpbuf, 1);
        fprintf(stderr, "Longjump failed, Exiting\n");
        exit(-1);
    }
    ifnode->next = ifnode->data.ifnode->thennode;
    ifnode->width = -1;         /* A flag for compute if extents */
    curr_node = endif;
    gInIf--;
}
```

9.26.2 parseCondnode

⟨*hypertex*⟩+≡

```
static void parseCondnode(void) {
    getToken();
    switch (token.type) {
      case Cond:
        curr_node->type = Cond;
        curr_node->data.text = allocString(token.id);
        break;
      case Haslisp:
      case Hasreturn:
      case Lastwindow:
      case Hasup:
        curr_node->type = token.type;
        break;
      case Boxcond:
        curr_node->type = Boxcond;
        curr_node->data.text = allocString(token.id);
        break;
      case Hasreturnto:
        parseHasreturnto();
        break;
      default:
        {
            char eb[128];
            tokenName(token.type);
            sprintf(eb, "Unexpected Token %s\n", eb);
            tpderror(eb, HTCONDNODE);
        }
        break;
    }
}
```

9.26.3 parseHasreturnto

⟨*hypertex*⟩+≡

```
static void parseHasreturnto(void) {
    TextNode *hrt = curr_node, *arg_node = allocNode();
    curr_node->type = Hasreturnto;
    curr_node = arg_node;
    getExpectedToken(Lbrace);
    parseHyperDoc();
    curr_node->type = Endarg;
    hrt->data.node = arg_node;
    curr_node = hrt;
}
```

9.26.4 parseNewcond

⟨*hypertex*⟩+≡

```
void parseNewcond(void) {
    char label[256];
    getExpectedToken(Lbrace);
    getExpectedToken(Unkeyword);
    strcpy(label, token.id);
    getExpectedToken(Rbrace);
    insertCond(label, "0");
    curr_node->type = Noop;
}
```

9.26.5 parseSetcond

⟨*hypertex*⟩+≡

```
void parseSetcond(void) {
    char label[256], cond[256];
    getExpectedToken(Lbrace);
    getExpectedToken(Cond);
    strcpy(label, token.id);
    getExpectedToken(Rbrace);
    getExpectedToken(Lbrace);
    getExpectedToken(Word);
    strcpy(cond, token.id);
    getExpectedToken(Rbrace);
    changeCond(label, cond);
    curr_node->type = Noop;
}
```

9.26.6 parseBeginItems

⟨*hypertex*⟩+≡

```
void parseBeginItems(void) {
    TextNode *bi = curr_node;
    /*
     * This procedure parses a begin item. It sets the current
     * node and sees if there is an optional argument for the itemspace
     */
    bi->type = token.type;
    getToken();
    if (token.type == Lsquarebrace) {
        bi->data.node = allocNode();
        curr_node = bi->data.node;
        gInOptional++;
        parseHyperDoc();
        gInOptional--;
        curr_node->type = Enddescription;
        if (token.type != Rsquarebrace) {
          fprintf(stderr, "(HyperDoc) Optional arguments must end with ].\n");
          printNextTenTokens();
          printPageAndFilename();
          jump();
        }
        curr_node = bi;
    }
    else
        ungetToken();
    gInItems++;
}
```

9.26.7 parseItem

⟨*hypertex*⟩+≡

```
void parseItem(void) {
    if (!gInItems) {
        fprintf(stderr, "\\item found outside an items environment\n");
        printPageAndFilename();
        printNextTenTokens();
        jump();
    }
    curr_node->type = Item;
    getToken();
    if (token.type == Lsquarebrace) {
        /* I should parse the optional argument */
        curr_node->next = allocNode();
        curr_node = curr_node->next;
        curr_node->type = Description;
        curr_node->next = allocNode();
        curr_node = curr_node->next;
        gInOptional++;
        parseHyperDoc();
        gInOptional--;
        curr_node->type = Enddescription;
        if (token.type != Rsquarebrace) {
          fprintf(stderr, "(HyperDoc) Optional arguments must end with ].\n");
          printNextTenTokens();
          printPageAndFilename();
          jump();
        }
    }
    else {
        ungetToken();
    }
}
```

9.26.8 parseMitem

⟨*hypertex*⟩+≡

```
void parseMitem(void) {
    if (!gInItems) {
        fprintf(stderr, "\\mitem found outside an items environment\n");
        printPageAndFilename();
        printNextTenTokens();
        jump();
    }
    curr_node->type = Mitem;
}
```

9.26.9 parseVerbatim

⟨*hypertex*⟩+≡

```
void parseVerbatim(int type) {
    int size = 0, c;
    char *end_string, *vb = vbuf, *es;
    curr_node->type = type;
    if (token.id[-1])
        curr_node->space = 1;
    if (type == Spadsrctxt) {
        es = end_string = "\n\\end{spadsrc}";
    }
    else if (type == Math)
        es = end_string = "$";
    else
        es = end_string = "\\end{verbatim}";
    while ((c = getChar()) != EOF) {
        resizeVbuf();
        size++;
        if (c == '\n') {
            new_verb_node();
            continue;
        }
        *vb++ = c;
        if (*es++ != c)
            es = end_string;
        if (!*es)
            break;
    }
    if (c == EOF) {
        fprintf(stderr, "parseVerbatim: Unexpected EOF found\n");
        longjmp(jmpbuf, 1);
    }
    resizeVbuf();
    if (*end_string == '\n')
        es = end_string + 1;
    else
        es = end_string;
    vbuf[size - strlen(es)] = '\0';
    if (*vbuf) {
        curr_node->data.text = allocString(vbuf);
        curr_node->next = allocNode();
        curr_node = curr_node->next;
    }
    if (type == Spadsrctxt)
        curr_node->type = Endspadsrc;
```

```
    else if (type == Math)
        curr_node->type = Endmath;
    else
        curr_node->type = Endverbatim;
}
```

9.26.10 parseInputPix

⟨*hypertex*⟩+≡

```
void parseInputPix(void) {
    TextNode *pixnode;
    char *filename;
    pixnode = curr_node;
    pixnode->type = token.type;
    pixnode->space = token.id[-1];
    pixnode->width = -1;
    getExpectedToken(Lbrace);
    filename = getInputString();
    pixnode->data.text = allocString(filename);
    curr_node = pixnode;
    if (pixnode->type == Inputimage) {
        char f[256];
        char *p;
        if ((gXDisplay && DisplayPlanes(gXDisplay, gXScreenNumber) == 1) ||
             gSwitch_to_mono ==1) {
           pixnode->type = Inputbitmap;
           strcpy(f, pixnode->data.text);
           strcat(f, ".bm");
           p=pixnode->data.text;
           pixnode->data.text = allocString(f);
           free(p);
        }
        else {
            pixnode->type = Inputpixmap;
            strcpy(f, pixnode->data.text);
            strcat(f, ".xpm");
            p=pixnode->data.text;
            pixnode->data.text = allocString(f);
            free(p);
        }
    }
}
```

9.26.11 parseCenterline

⟨*hypertex*⟩+≡

```
void parseCenterline(void) {
    curr_node->type = token.type;
    curr_node->space = token.id[-1];
    curr_node->width = -1;
    curr_node->next = allocNode();
    curr_node = curr_node->next;
    getExpectedToken(Lbrace);
    parseHyperDoc();
    if (token.type != Rbrace) {
        curr_node->type = Noop;
        fprintf(stderr, "(HyperdDoc) \\centerline was expecting a }\n");
        printPageAndFilename();
        printNextTenTokens();
        longjmp(jmpbuf, 1);
    }
    curr_node->type = Endcenter;
}
```

9.26.12 parseCommand

⟨*hypertex*⟩+≡

```
void parseCommand(void) {
    TextNode *link_node, *save_node, *arg_node;
    gInButton++;
    if (gParserMode == SimpleMode) {
        curr_node->type = Noop;
        fprintf(stderr, "Parser Error token %s unexpected\n",
                token_table[token.type]);
        longjmp(jmpbuf, 1);
    }
    gStringValueOk = 1;
    /* set the values for the current node */
    curr_node->type = token.type;
    curr_node->space = token.id[-1];
    /* now parse for the label */
    link_node = curr_node;
    curr_node->next = allocNode();
    curr_node = curr_node->next;
    getExpectedToken(Lbrace);
    parseHyperDoc();
    curr_node->type = Endbutton;
    save_node = curr_node;
    arg_node = allocNode();
    curr_node = arg_node;
    getExpectedToken(Lbrace);
    parseHyperDoc();
    curr_node->type = Endarg;
    link_node->link = makeLinkWindow(arg_node, link_node->type, 0);
    gStringValueOk = 0;
    curr_node = save_node;
    gInButton--;
}
```

9.26.13 parseButton

⟨*hypertex*⟩+≡

```
void parseButton(void) {
    TextNode *link_node, *save_node;
    gInButton++;
    if (gParserMode == SimpleMode) {
        curr_node->type = Noop;
        fprintf(stderr, "Parser Error token %s unexpected\n",
                token_table[token.type]);
        longjmp(jmpbuf, 1);
    }
    /* fill the node */
    curr_node->type = token.type;
    curr_node->space = token.id[-1];
    /* the save the current node for creating the link and stuff */
    link_node = curr_node;
    /* then parse the label */
    curr_node->next = allocNode();
    curr_node = curr_node->next;
    getExpectedToken(Lbrace);
    parseHyperDoc();
    curr_node->type = Endbutton;
    /* now try to get the argument node */
    save_node = curr_node;
    getExpectedToken(Lbrace);
    save_node->data.node = allocNode();
    curr_node = save_node->data.node;
    parseHyperDoc();
    curr_node->type = Endarg;
    /*
     * buffer[0] = '\0'; printToString(arg_node, buffer + 1);
     */
    link_node->link =
        makeLinkWindow(save_node->data.node, link_node->type, 0);
    curr_node = save_node;
    gInButton--;
}
```

9.26.14 parseSpadcommand

⟨*hypertex*⟩+≡

```
void parseSpadcommand(TextNode *spad_node) {
    example_number++;
    gInButton++;
    spad_node->type = token.type;
    spad_node->space = token.id[-1];
    getExpectedToken(Lbrace);
    cur_spadcom = curr_node;
    spad_node->next = allocNode();
    curr_node = spad_node->next;
    parseHyperDoc();
    curr_node->type = Endspadcommand;
    cur_spadcom = NULL;
    spad_node->link = makeLinkWindow(spad_node->next, spad_node->type, 1);
    gInButton--;
}
```

9.26.15 parseSpadsrc

⟨*hypertex*⟩+≡

```
void parseSpadsrc(TextNode *spad_node) {
    char buf[512], *c = buf;
    int ch, start_opts = 0;
    /*TextNode *node = NULL;*/
    example_number++;
    gInButton++;
    gInSpadsrc++;
    spad_node->type = Spadsrc;
    spad_node->space = token.id[-1];
    cur_spadcom = curr_node;
    spad_node->next = allocNode();
    curr_node = spad_node->next;
    do {
        ch = getChar();
        if (ch == ']')
            start_opts = 0;
        if (start_opts)
            *c++ = ch;
        if (ch == '[')
            start_opts = 1;
    } while (ch != '\n');
    *c = '\0';
    parseVerbatim(Spadsrctxt);
    parseFromString(buf);
    curr_node->type = Endspadsrc;
    cur_spadcom = NULL;
    spad_node->link = makeLinkWindow(spad_node->next, Spadsrc, 1);
    gInButton--;
    gInSpadsrc--;
}
```

9.26.16 parseEnv

⟨*hypertex*⟩+≡

```
void parseEnv(TextNode *node) {
    char *env;
    char buff[256];
    char *buff_pntr = &buff[1];
    int  noEnv = 0;
    getExpectedToken(Lbrace);
    getExpectedToken(Word);
    env = getenv(token.id);
    if (env == NULL) {
        /** The environment variable was not found **/
        fprintf(stderr,
           "(HyperDoc) Warning: environment variable \'%s\' was not found.\n",
           token.id);
        env = halloc(1, "string");
        env[0] = '\0';
        noEnv = 1;
    }
    buff[0] = token.id[-1];
    strcpy(buff_pntr, env);
    if (noEnv)
        free(env);
    node->data.text = allocString(buff_pntr);
    node->type = Word;
    getExpectedToken(Rbrace);
}
```

9.26.17 parseValue1

This parseValue routine accepts an empty `{}` but makes it a zero instead of a one. Thus `\indent{}` is equivelant to `\indent{0}`.

⟨*hypertex*⟩+≡

```
void parseValue1(void) {
    TextNode *value_node, *ocn = curr_node;
    char *s;
    curr_node->type = token.type;
    curr_node->space = token.id[-1];
    value_node = allocNode();
    value_node->type = Word;
    curr_node->data.node = value_node;
    getExpectedToken(Lbrace);
    s = getInputString();
    if (!isNumber(s)) {
        fprintf(stderr,
           "Parser Error: parse for value was expecting a numeric value\n");
        strcpy(value_node->data.text, "0");
    }
    else {
        value_node->data.text = allocString(s);
    }
    curr_node = ocn;
}
```

9.26.18 parseValue2

This command accepts an empty argument command. Thus `\space{}` is equivelant `\space{1}`

⟨*hypertex*⟩+≡

```
void parseValue2(void) {
    TextNode *value_node, *ocn = curr_node;
    char *s;
    curr_node->type = token.type;
    curr_node->space = token.id[-1];
    value_node = allocNode();
    value_node->type = Word;
    curr_node->data.node = value_node;
    getExpectedToken(Lbrace);
    s = getInputString();
    if (!isNumber(s)) {
        fprintf(stderr,
           "Parser Error: parse for value was expecting a numeric value\n");
        strcpy(value_node->data.text, "1");
    }
    else {
        value_node->data.text = allocString(s);
    }
    curr_node = ocn;
}
```

9.26.19 parseTable

Parse a `\table` command.

⟨*hypertex*⟩+≡

```
void parseTable(void) {
    TextNode *tn = curr_node;
    if (gParserMode != AllMode) {
        curr_node->type = Noop;
        fprintf(stderr, "Parser Error token %s unexpected\n",
                token_table[token.type]);
        longjmp(jmpbuf, 1);
    }
    curr_node->type = Table;
    getExpectedToken(Lbrace);
    curr_node->next = allocNode();
    curr_node = curr_node->next;
    getToken();
    if (token.type == Lbrace) {
        while (token.type != Rbrace) {
            curr_node->type = Tableitem;
            curr_node->next = allocNode();
            curr_node = curr_node->next;
            parseHyperDoc();
            curr_node->type = Endtableitem;
            curr_node->next = allocNode();
            curr_node = curr_node->next;
            getToken();
        }
        curr_node->type = Endtable;
    }
    else {                      /* a patch for SG for empty tables */
        if (token.type != Rbrace) {
            tokenName(token.type);
            fprintf(stderr,
                    "Unexpected Token %s found while parsing a table\n",
                    ebuffer);
            printPageAndFilename();
            jump();
        }
        tn->type = Noop;
        tn->next = NULL;
        free(curr_node);
        curr_node = tn;
    }
}
```

9.26.20 parseBox

⟨*hypertex*⟩+≡

```
void parseBox(void) {
    curr_node->type = token.type;
    curr_node->space = token.id[-1];
    curr_node->width = -1;
    curr_node->next = allocNode();
    curr_node = curr_node->next;
    getExpectedToken(Lbrace);
    parseHyperDoc();
    curr_node->type = Endbox;
}
```

9.26.21 parseMbox

⟨*hypertex*⟩+≡

```
void parseMbox(void) {
    curr_node->type = token.type;
    curr_node->space = token.id[-1];
    curr_node->width = -1;
    curr_node->next = allocNode();
    curr_node = curr_node->next;
    getExpectedToken(Lbrace);
    parseHyperDoc();
    curr_node->type = Endbox;
}
```

9.26.22 parseFree

⟨*hypertex*⟩+≡

```
void parseFree(void) {
    TextNode *freeNode = curr_node;
    curr_node->type = token.type;
    curr_node->space = token.id[-1];
    curr_node->width = -1;
    curr_node->data.node = allocNode();
    curr_node = curr_node->data.node;
    getExpectedToken(Lbrace);
    parseHyperDoc();
    curr_node->type = Endarg;
    curr_node = freeNode;
}
```

9.26.23 parseHelp

⟨*hypertex*⟩+≡

```
void parseHelp(void) {
    curr_node->type = Noop;
    getToken();
    if (token.type != Lbrace) {
        tokenName(token.type);
        fprintf(stderr,"\\helppage was expecting a { and not a %s\n", ebuffer);
        printPageAndFilename();
        jump();
    }
   /* before we clobber this pointer we better free the contents
      (cf. allocPage) */
    free(gPageBeingParsed->helppage);
    gPageBeingParsed->helppage = allocString(getInputString());
    if (token.type != Rbrace) {
        tokenName(token.type);
        fprintf(stderr, "\\helppage was expecting a } and not a %s\n",
                ebuffer);
        printPageAndFilename();
        jump();
    }
}
```

9.27 Reading bitmaps

9.27.1 HTReadBitmapFile

This file was produced by J.M. Wiley with some help from the bitmap editor routine. It reads in a bitmap file, and calls XCreatePixmapFromBitmapData to transform it into a Pixmap. He did this because the routine XReadBitmapFile does not seeem to work too well (whatever that means).

⟨*hypertex*⟩+≡

```
XImage *HTReadBitmapFile(Display *display,int screen,char * filename,
                         int *width, int *height) {
    XImage *image;
    FILE *fd;
    char Line[256], Buff[256];
    int num_chars;
    char *ptr;
    int rch;
    int version;
    int padding, chars_line, file_chars_line, file_chars;
    int bytes;
    int x_hot, y_hot;
    image = XCreateImage(display, DefaultVisual(display, screen), 1,
                         XYBitmap, 0, NULL, 0, 0, 8, 0);
    (image)->byte_order = LSBFirst;     /* byte_order    */
    (image)->bitmap_unit = 8;   /* bitmap-unit   */
    (image)->bitmap_bit_order = LSBFirst;       /* bitmap-bit-order */
    if (!(fd = zzopen(filename, "r"))) {
        fprintf(stderr, "ReadBitmapFile: File >%s< not found\n", filename);
        exit(-1);
    }
    /*
     * Once it is open, lets get the width and height
     */
    if ((readWandH(fd,(unsigned int *)width,(unsigned int *) height)) < 0) {
        fprintf(stderr, "ReadBitmapFile: Bad file format in %s\n", filename);
        exit(-1);
    }
    /*
     * Now get the next line, and see if it is hot spots or bits
     */
    if (fgets(Line, MAXLINE, fd) == NULL) {
        fprintf(stderr, "ReadBitmapFile: Bad file format in %s\n", filename);
        exit(-1);
    }
    /*
```

```
 * Now check the first character to see if it is a # or an s
 */
if (Line[0] == '#') {
  if ((readHot(fd, Line, &x_hot, &y_hot)) < 0) {
    fprintf(stderr, "ReadBitmapFile: Bad file format in %s\n", filename);
    exit(-1);
  }
}
(image)->width = *width;
(image)->height = *height;
/*
 * figure out what version
 */
if (sscanf(Line, "static short %s = {", Buff) == 1)
    version = 10;
else if (sscanf(Line, "static unsigned char %s = {", Buff) == 1)
    version = 11;
else if (sscanf(Line, "static char %s = {", Buff) == 1)
    version = 11;
else {
    fprintf(stderr, "ReadBitmapFile: Bad file format in %s\n", filename);
    exit(-1);
}
padding = 0;
if ((*width % 16) && ((*width % 16) < 9) && (version == 10))
    padding = 1;
(image)->bytes_per_line = chars_line = (*width + 7) / 8;
file_chars_line = chars_line + padding;
num_chars = chars_line * (*height);
file_chars = file_chars_line * (*height);
(image)->data = (char *) halloc((image)->bytes_per_line * (image)->height,
                                "Read Pixmap--Image data");
/*
 * Since we are just holding the first line of the declaration, we can
 * just start reading from fd
 */
if (version == 10)
  for (bytes = 0, ptr = (image)->data; bytes < file_chars; (bytes += 2)) {
    if (fscanf(fd, " 0x%x%*[,}]%*[ \n]", &rch) != 1) {
      fprintf(stderr, "ReadBitmapFile: Bad file format in %s\n", filename);
      exit(-1);
    }
    *(ptr++) = rch & 0xff;
    if (!padding || ((bytes + 2) % file_chars_line))
      *(ptr++) = rch >> 8;
}
```

```
    else
      for (bytes=0, ptr = (image)->data; bytes < file_chars; bytes++, ptr++) {
        if (fscanf(fd, " 0x%x%*[,}]%*[ \n]", &rch) != 1) {
          fprintf(stderr, "ReadBitmapFile: Bad file format in %s\n", filename);
          exit(-1);
        }
        *ptr = rch;
    }
    fclose(fd);
    return image;
}
```

9.27.2 readHot

⟨*hypertex*⟩+≡

```
static int readHot(FILE *fd,char Line[],int *x_hot,int *y_hot) {
    char Buff[256];
    /*
     * Works much the same as get width and height, just new variables
     */
    if (sscanf(Line, "#define %s %d", Buff, x_hot) != 2)
        return -1;
    if (fgets(Line, MAXLINE, fd) == NULL)
        return -1;
    if (sscanf(Line, "#define %s %d", Buff, y_hot) != 2)
        return -1;
    if (fgets(Line, MAXLINE, fd) == NULL)
        return -1;
    return 1;
}
```

9.27.3 readWandH

⟨*hypertex*⟩+≡

```
static int readWandH(FILE *fd,unsigned int *width,unsigned int *height) {
    char Line[256], Buff[256];
    if (fgets(Line, MAXLINE, fd) == NULL)
        return -1;
    /*
     * Once we have the line, scan it for the width
     */
    if (sscanf(Line, "#define %s %d", Buff, width) != 2)
        return -1;
    /*
     * Hopefully we have the width, now get the height the same way
     */
    if (fgets(Line, MAXLINE, fd) == NULL)
        return -1;
    /*
     * Once we have the line, scan it for the height
     */
    if (sscanf(Line, "#define %s %d", Buff, height) != 2)
        return -1;
    return 1;
}
```

9.27.4 insertImageStruct

Read a bitmap file into memory.

⟨*hypertex*⟩+≡

```
ImageStruct *insertImageStruct(char *filename) {
    int bm_width, bm_height;
    XImage *im;
    ImageStruct *image;
    if (*filename == ' ')
        filename++;
    if ((image=(ImageStruct *) hashFind(&gImageHashTable, filename)) == NULL) {
        im = HTReadBitmapFile(gXDisplay, gXScreenNumber, filename,
                              &bm_width, &bm_height);
        /*
         * now add the image to the gImageHashTable
         */
        image = (ImageStruct *) halloc(sizeof(ImageStruct), "ImageStruct");
        image->image.xi = im;
        image->width = image->image.xi->width;
        image->height = image->image.xi->height;
        image->filename = (char *) halloc(sizeof(char) * strlen(filename) +1,
                                          "insert_image--filename");
        /* strcpy(image->filename, filename); */
        sprintf(image->filename, "%s", filename);
        hashInsert(&gImageHashTable,(char *) image, image->filename);
    }
    return image;
}
```

9.28 Scrollbar handling routines

The scrollbar is displayed on the side of the HyperDoc display, if needed. It is composed of four windows

- fScrollUpWindow – the arrowed box at the top of the scrollbar. Scrolls the window up a line at a time.
- fScrollDownWindow – Located at the bottom of the window, it is used to scroll down a single line at a time.
- scrollbar – this is the window which does the variable scrolling. It houses the actual scroller.
- scroller – This is the scroller inside the scroll bar.

The procedure below, makes all these windows, and also makes three bitmaps,

- sup – The up arrow for the fScrollUpWindow.
- sdown – the down arrow for the fScrollDownWindow.
- scroller – the scroller stipple.

It then fills the window with the proper Pixmap background.

The scrollbar and scroller works as follows. The size of the scroller is calculated as

```
size of scroller            size of visible text
-----------------  ===  ------------------------------  .
size of scrollbar       size of whole scrolling region
```

The top of the scroller shows the relative position in the page of the top of the scrolling region. This way the user knows how far down the page he or she has moved. When the user clicks in the scrollbar, the center of the scroller, if possible, is placed at the point of the click.

See the routines

- showScrollBars – to see how the scroll bars are actually realized.
- moveScroller – to see how the scroller is moved when the user scrolls

9.28.1 makeScrollBarWindows

⟨*hypertex*⟩+≡

```
void makeScrollBarWindows(void) {
    XSetWindowAttributes at;
    at.cursor = gActiveCursor;
    at.event_mask = ButtonPress;
    /** create the bitmaps **/
    if (supwidth != sdown_width || supheight != sdown_height) {
        fprintf(stderr,
      "Scrollbar error, up and down pointers must have the same dimensions\n");
        exit(-1);
    }
    if (sup == 0)
      sup =
        XCreatePixmapFromBitmapData(gXDisplay,
          RootWindow(gXDisplay, gXScreenNumber), sup_bits, supwidth, supheight,
          FORECOLOR, BACKCOLOR, DefaultDepth(gXDisplay, gXScreenNumber));
    if (sdown == 0)
      sdown =
        XCreatePixmapFromBitmapData(gXDisplay,
          RootWindow(gXDisplay, gXScreenNumber), sdown_bits, sdown_width,
          sdown_height, FORECOLOR, BACKCOLOR,
          DefaultDepth(gXDisplay, gXScreenNumber));
    sup_pressed =
       XCreatePixmapFromBitmapData(gXDisplay,
          RootWindow(gXDisplay, gXScreenNumber), sup3dpr_bits, sup3dpr_width,
          sup3dpr_height, FORECOLOR, BACKCOLOR,
          DefaultDepth(gXDisplay, gXScreenNumber));
    sdown_pressed =
       XCreatePixmapFromBitmapData(gXDisplay,
          RootWindow(gXDisplay, gXScreenNumber), sdown3dpr_bits,
          sdown3dpr_width, sdown3dpr_height, FORECOLOR, BACKCOLOR,
          DefaultDepth(gXDisplay, gXScreenNumber));
    gWindow->fScrollUpWindow =
       XCreateSimpleWindow(gXDisplay, gWindow->fMainWindow, 1, 1, supwidth,
         supheight, gWindow->border_width, gBorderColor, BACKCOLOR);
    gWindow->fScrollDownWindow =
       XCreateSimpleWindow(gXDisplay, gWindow->fMainWindow, 1, 1, sdown_width,
         sdown_height, gWindow->border_width, gBorderColor, BACKCOLOR);
    gWindow->scrollbar =
       XCreateSimpleWindow(gXDisplay, gWindow->fMainWindow, 1, 1, 1, 1,
         gWindow->border_width, gBorderColor, BACKCOLOR);
    gWindow->scroller =
       XCreateSimpleWindow(gXDisplay, gWindow->scrollbar, 1, 1, 1, 1, 0,
         gBorderColor, BACKCOLOR);
```

```
#ifdef DEBUG
    fprintf(stderr, "Changing Window Attributes in scrollbar.c #2\n");
#endif
    at.background_pixmap = sup;
    XChangeWindowAttributes(gXDisplay, gWindow->fScrollUpWindow,
                            CWBackPixmap | CWEventMask | CWCursor, &at);
    at.background_pixmap = sdown;
    XChangeWindowAttributes(gXDisplay, gWindow->fScrollDownWindow,
                            CWBackPixmap | CWEventMask | CWCursor, &at);
    XChangeWindowAttributes(gXDisplay, gWindow->scrollbar,
                            CWEventMask | CWCursor, &at);
    if (scroller == 0)
      scroller =
        XCreatePixmapFromBitmapData(gXDisplay,
          RootWindow(gXDisplay, gXScreenNumber), scroller_bits, scroller_width,
          scroller_height, FORECOLOR, BACKCOLOR,
          DefaultDepth(gXDisplay, gXScreenNumber));
    if (scrollbar_pix == 0)
      scrollbar_pix =
        XCreatePixmapFromBitmapData(gXDisplay,
          RootWindow(gXDisplay, gXScreenNumber), scrollbar_pix_bits,
          scrollbar_pix_width, scrollbar_pix_height, FORECOLOR, BACKCOLOR,
          DefaultDepth(gXDisplay, gXScreenNumber));
    at.background_pixmap = scroller;
    XChangeWindowAttributes(gXDisplay, gWindow->scroller,
                            CWBackPixmap | CWCursor, &at);
    at.background_pixmap = scrollbar_pix;
    XChangeWindowAttributes(gXDisplay, gWindow->scrollbar,
                            CWBackPixmap, &at);
}
```

9.28.2 drawScroller3DEffects

⟨*hypertex*⟩+≡

```
static void drawScroller3DEffects(HDWindow * hdWindow, int x1, int y1,
                                  int x2, int y2) {
    XClearWindow(gXDisplay, hdWindow->scroller);
    /* draw right "black" line */
    XDrawLine(gXDisplay, hdWindow->scroller, hdWindow->fControlGC,
              x2 - 3, y1 + 2, x2 - 3, y2 - 3);
    /* draw bottom "black" line */
    XDrawLine(gXDisplay, hdWindow->scroller, hdWindow->fControlGC,
              x1 + 2, y2 - 3, x2 - 3, y2 - 3);
    /* flip foreground and background colors */
    XSetBackground(gXDisplay, hdWindow->fControlGC, gControlForegroundColor);
    XSetForeground(gXDisplay, hdWindow->fControlGC, gControlBackgroundColor);
    /* draw top "white" line */
    XDrawLine(gXDisplay, hdWindow->scroller, hdWindow->fControlGC,
              x1 + 2, y1 + 2, x2 - 3, y1 + 2);
    /* draw left "white" line */
    XDrawLine(gXDisplay, hdWindow->scroller, hdWindow->fControlGC,
              x1 + 2, y1 + 2, x1 + 2, y2 - 3);
    /* reset colors */
    XSetBackground(gXDisplay, hdWindow->fControlGC, gControlBackgroundColor);
    XSetForeground(gXDisplay, hdWindow->fControlGC, gControlForegroundColor);
}
```

9.28.3 showScrollBars

⟨*hypertex*⟩+≡

```
void showScrollBars(HDWindow * hdWindow) {
    XWindowChanges wc;
    /*int src_x = 0, src_y = 0;*/
    /*unsigned int width = supwidth, height = supheight;*/
    /*int dest_x = 0, dest_y = 0;*/
    /* see if we even need scroll bars */
    if (hdWindow->page->scrolling->height <= hdWindow->scrollheight)
        return;
    wc.x = hdWindow->scrollx;
    wc.y = hdWindow->scrollupy;
    wc.height = supheight;
    wc.width = supwidth;
    XConfigureWindow(gXDisplay, hdWindow->fScrollUpWindow, CWX | CWY | CWHeight
                     | CWWidth, &wc);
    wc.y = hdWindow->scrolldowny;
    XConfigureWindow(gXDisplay, hdWindow->fScrollDownWindow,
                     CWX | CWY | CWHeight | CWWidth,
                     &wc);
    wc.height = hdWindow->fScrollBarHeight;
    wc.y = hdWindow->scrollbary;
    XConfigureWindow(gXDisplay, hdWindow->scrollbar,
                     CWX | CWY | CWHeight | CWWidth,
                     &wc);
    wc.x = 0;
    wc.y = hdWindow->fScrollerTopPos;
    wc.width = supwidth;
    wc.height = hdWindow->fScrollerHeight;
    XConfigureWindow(gXDisplay, hdWindow->scroller,
                     CWX | CWY | CWHeight | CWWidth,
                     &wc);
    /*
     * Now we map the windows, since the bitmaps are the backgrounds for the
     * windows, we need to worry about redrawing them.
     */
    XMapWindow(gXDisplay, hdWindow->fScrollUpWindow);
    XMapWindow(gXDisplay, hdWindow->fScrollDownWindow);
    XMapWindow(gXDisplay, hdWindow->scrollbar);
    XMapWindow(gXDisplay, hdWindow->scroller);
    drawScroller3DEffects(hdWindow, 0, 0, wc.width, wc.height);
}


/*********************************************************************
```

```
Moves the scroller to its proper place within the scrollbar. It
calculates how far down the page we are, and then moves the scroller
accordingly

**************************************************************************/
```

9.28.4 moveScroller

Moves the scroller to it's proper place.

⟨*hypertex*⟩+≡

```
void moveScroller(HDWindow * hdWindow) {
    XWindowChanges wc;
    int t = (int) (hdWindow->fScrollBarHeight * (-hdWindow->page->scroll_off));
    hdWindow->fScrollerTopPos = (int) (t / hdWindow->page->scrolling->height);
    wc.x = 0;
    wc.y = hdWindow->fScrollerTopPos;
    wc.width = supwidth;
    wc.height = hdWindow->fScrollerHeight;
    XConfigureWindow(gXDisplay, hdWindow->scroller,
                     CWX | CWY | CWHeight | CWWidth,
                     &wc);
    drawScroller3DEffects(hdWindow, 0, 0, wc.width, wc.height);
}
```

9.28.5 drawScrollLines

Checks the pageFlags to see if we need a top, or a bottom line. These are the horizontal lines framing a scrolling region when the scrolling region is not the entire window.

⟨*hypertex*⟩+≡

```
void drawScrollLines(void) {
    if (!(gWindow->page->pageFlags & NOLINES)) {
        lineTopGroup();
        if (gWindow->page->header->height) {
            XDrawLine(gXDisplay, gWindow->fMainWindow, gWindow->fStandardGC,
                      0,
                      gWindow->page->top_scroll_margin -
                        tophalf(gWindow->border_width) -
                        2 * scroll_top_margin,
                      gWindow->scrollwidth,
                      gWindow->page->top_scroll_margin -
                        tophalf(gWindow->border_width) -
                        2 * scroll_top_margin);
        }
        if (gWindow->page->footer->height) {
            XDrawLine(gXDisplay, gWindow->fMainWindow, gWindow->fStandardGC,
                      0,
                      gWindow->page->bot_scroll_margin +
                        bothalf(gWindow->border_width) - 1,
                      gWindow->scrollwidth,
                      gWindow->page->bot_scroll_margin +
                        bothalf(gWindow->border_width) - 1);
        }
        popGroupStack();
    }
}
```

9.28.6 calculateScrollBarMeasures

Calculates all the measures for the scrollbars.

⟨*hypertex*⟩+≡

```
void calculateScrollBarMeasures(void) {
    int t;
    /*
     * The scrollhieght is the height of the scrolling region visible in the
     * HT window. Notice how it is a multiple of line height. This was needed
     * to make everything scroll nicely.
     */
    gWindow->scrollheight = gWindow->page->bot_scroll_margin -
        gWindow->page->top_scroll_margin - scroll_top_margin;
    gWindow->scrollheight = gWindow->scrollheight -
        gWindow->scrollheight % line_height;
    /*
     * Now do a quick check to see if I really need a scroll bar, and if not,
     * just return right away
     */
    if (gWindow->scrollheight >= gWindow->page->scrolling->height) {
        gWindow->page->scroll_off = 0;
        return;
    }
    /*
     * The height of the scrollbar region, extends form the top page margin
     * all the way to the bottom, excluding the room needed for the up and
     * down windows
     */
    gWindow->fScrollBarHeight = gWindow->page->bot_scroll_margin -
        gWindow->page->top_scroll_margin - 2 * supheight -
        2 * gWindow->border_width;
    gWindow->scrollupy =
      gWindow->page->top_scroll_margin - gWindow->border_width;
    gWindow->scrollupy -= 2 * scroll_top_margin;
    gWindow->scrolldowny = gWindow->page->bot_scroll_margin
        - supheight - gWindow->border_width;
    gWindow->scrollbary =
      gWindow->scrollupy + supheight + gWindow->border_width;
    gWindow->scrollx = gWindow->width - supwidth - gWindow->border_width;
    /*
     * the scroller height is calculated from the following formula
     *
     * fScrollerHeight                    scrollheight --------------       ==
     * --------- ------------- fScrollBarHeight
     * page->scrolling_height
```

```
     *
     */
     /** possible integer error correction **/
    gWindow->fScrollerHeight = 1 + 2 * scroll_top_margin +
       (int) (gWindow->fScrollBarHeight *
       gWindow->scrollheight / gWindow->page->scrolling->height);
    /*
     * Check the scroll offset, to see if it is too Large
     */
    if (-(gWindow->page->scroll_off) >
        (gWindow->page->scrolling->height - gWindow->scrollheight))
        gWindow->page->scroll_off =
            -(gWindow->page->scrolling->height - gWindow->scrollheight);
    /*
     * Then move the top of the scroller to it's proper position
     */
    gWindow->fScrollBarHeight += 2 * scroll_top_margin;
    t = (int) (gWindow->fScrollBarHeight * (-gWindow->page->scroll_off));
    gWindow->fScrollerTopPos = (int) (t / (gWindow->page->scrolling->height));
}
```

9.28.7 linkScrollBars

⟨*hypertex*⟩+≡

```
void linkScrollBars(void) {
    HyperLink *uplink = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink");
    HyperLink *downlink = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink");
    HyperLink *barlink = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink");
    uplink->win = gWindow->fScrollUpWindow;
    downlink->win = gWindow->fScrollDownWindow;
    barlink->win = gWindow->scrollbar;
    uplink->type = Scrollupbutton;
    downlink->type = Scrolldownbutton;
    barlink->type = Scrollbar;
    barlink->x = barlink->y = 0;
    uplink->x = uplink->y = 0;
    downlink->x = downlink->y = 0;
    uplink->reference.node = NULL;
    downlink->reference.node = NULL;
    hashInsert(gLinkHashTable, (char *)uplink,(char *) &uplink->win);
    hashInsert(gLinkHashTable, (char *)barlink,(char *) &barlink->win);
    hashInsert(gLinkHashTable, (char *)downlink,(char *) &downlink->win);
}
```

9.28.8 scrollUp

⟨*hypertex*⟩+≡

```
void scrollUp(void) {
    if (gWindow->page->scroll_off == 0);       /* BeepAtTheUser(); */// The
                                                 * beeping annoyed me. RSS */
    else {
        changeWindowBackgroundPixmap(gWindow->fScrollUpWindow, sup_pressed);
        gWindow->page->scroll_off += line_height;      /* Scroll a line */
        if (gWindow->page->scroll_off > 0)
            gWindow->page->scroll_off = 0;
        XCopyArea(gXDisplay, gWindow->fScrollWindow, gWindow->fScrollWindow,
          gWindow->fStandardGC, 0, 0, gWindow->scrollwidth,
          gWindow->scrollheight - line_height + 1, 0, line_height);
        XClearArea(gXDisplay, gWindow->fScrollWindow, 0, 0,
                   gWindow->scrollwidth, line_height, False);
        scrollPage(gWindow->page);
        changeWindowBackgroundPixmap(gWindow->fScrollUpWindow, sup);
    }
}
```

9.28.9 scrollUpPage

⟨*hypertex*⟩+≡

```
void scrollUpPage(void) {
    if (gWindow->page->scroll_off == 0);       /* BeepAtTheUser(); */
    else {
        /* Scroll a page */
        gWindow->page->scroll_off += ch(gWindow->scrollheight) - line_height;
        if (gWindow->page->scroll_off > 0)
            gWindow->page->scroll_off = 0;
        XClearWindow(gXDisplay, gWindow->fScrollWindow);
        scrollPage(gWindow->page);
    }
}
```

9.28.10 scrollToFirstPage

⟨*hypertex*⟩+≡

```
void scrollToFirstPage(void) {
    if (gWindow->page->scroll_off == 0);       /* BeepAtTheUser(); */
    else {
        gWindow->page->scroll_off = 0;
        XClearWindow(gXDisplay, gWindow->fScrollWindow);
        scrollPage(gWindow->page);
    }
}
```

9.28.11 scrollDown

⟨*hypertex*⟩+≡

```
void scrollDown(void) {
    if (-(gWindow->page->scroll_off) >=
        (gWindow->page->scrolling->height - gWindow->scrollheight)) {
        ;                       /* BeepAtTheUser(); */
    }
    else {
      changeWindowBackgroundPixmap(gWindow->fScrollDownWindow,sdown_pressed);
      gWindow->page->scroll_off -= line_height;      /* Scroll a line */
      XCopyArea(gXDisplay, gWindow->fScrollWindow, gWindow->fScrollWindow,
        gWindow->fStandardGC, 0, line_height, gWindow->scrollwidth,
        gWindow->scrollheight - line_height + 1, 0, 0);
      XClearArea(gXDisplay, gWindow->fScrollWindow, 0,
        gWindow->scrollheight - line_height, gWindow->scrollwidth,
        line_height, False);
      scrollPage(gWindow->page);
      changeWindowBackgroundPixmap(gWindow->fScrollDownWindow, sdown);
    }
}
```

9.28.12 scrollDownPage

⟨*hypertex*⟩+≡

```
void scrollDownPage(void) {
    if (gWindow->page->scrolling == NULL || (-(gWindow->page->scroll_off) >=
            (gWindow->page->scrolling->height - gWindow->scrollheight))) {
        ;                       /* BeepAtTheUser(); */
    }
    else {
        gWindow->page->scroll_off -= ch(gWindow->scrollheight) - line_height;
        if (-(gWindow->page->scroll_off) >
            (gWindow->page->scrolling->height - gWindow->scrollheight))
            gWindow->page->scroll_off = -
                (gWindow->page->scrolling->height - gWindow->scrollheight);
        XClearWindow(gXDisplay, gWindow->fScrollWindow);
        scrollPage(gWindow->page);
    }
}
```

9.28.13 scrollScroller

This routine checks to see where in the window the button press occured. It then tries to move the scroller so that the top of the scroller is at the spot of the event

⟨*hypertex*⟩+≡

```
void scrollScroller(XButtonEvent * event) {
    int y = event->y;
    int top = y;
    if (top < 0) {
        top = 0;
        if (gWindow->fScrollerTopPos == 0)
            return;
        gWindow->page->scroll_off = 0;
    }
    else if ((top + gWindow->fScrollerHeight) > gWindow->fScrollBarHeight) {
        top = gWindow->fScrollBarHeight - gWindow->fScrollerHeight;
        if (top == gWindow->fScrollerTopPos)
            return;
        gWindow->page->scroll_off =
            -(gWindow->page->scrolling->height - gWindow->scrollheight);
        gWindow->page->scroll_off -= gWindow->page->scroll_off % line_height;
    }
    else {                      /** top is in an ok spot **/
        int t;
        t = -(gWindow->page->scrolling->height) * top;
        t = t / (gWindow->fScrollBarHeight);
        if (gWindow->page->scroll_off == (t -= t % line_height))
            return;
        gWindow->page->scroll_off = t;
        gWindow->fScrollerTopPos = top;
    }
    XClearWindow(gXDisplay, gWindow->fScrollWindow);
    scrollPage(gWindow->page);
}
```

9.28.14 hideScrollBars

⟨*hypertex*⟩+≡
```
void hideScrollBars(HDWindow * hdWindow) {
    XUnmapWindow(gXDisplay, hdWindow->fScrollDownWindow);
    XUnmapWindow(gXDisplay, hdWindow->fScrollUpWindow);
    XUnmapWindow(gXDisplay, hdWindow->scrollbar);
    XUnmapWindow(gXDisplay, hdWindow->scroller);
}
```

9.28.15 getScrollBarMinimumSize

⟨*hypertex*⟩+≡
```
void getScrollBarMinimumSize(int *width, int *height) {
    (*width)  = sup_width + 4;
    (*height) = sup_height + sdown_height + 5;
}
```

9.28.16 ch

⟨*hypertex*⟩+≡
```
static int ch(int height) {
    int rem = height % line_height;
    if (rem == 0)
        return height;
    return height - rem + line_height;
}
```

9.28.17 changeWindowBackgroundPixmap

⟨*hypertex*⟩+≡
```
static void changeWindowBackgroundPixmap(Window window, Pixmap pixmap) {
    if (pixmap) {
        XSetWindowAttributes at;
        at.background_pixmap = pixmap;
        XChangeWindowAttributes(gXDisplay, window, CWBackPixmap, &at);
        XClearWindow(gXDisplay, window);
    }
}
```

9.29 Display text object

/* * Display the page whose extent has been computed, using the actual size of
* the window, and yOff to determine clipped areas */

9.29.1 showText

⟨*hypertex*⟩+≡

```
void showText(TextNode *node, int Ender) {
    for (; node != NULL; node = node->next) {
        switch (node->type) {
          case 0:
          case Beginitems:
          case Begintitems:
          case Bound:
          case Center:
          case Free:
          case HSpace:
          case Indent:
          case Indentrel:
          case Item:
          case Macro:
          case Mbox:
          case Newline:
          case Noop:
          case Par:
          case Pound:
          case Rbrace:
          case Space:
          case Tab:
          case Table:
          case Titem:
          case VSpace:
            break;
          case Dash:
          case Fi:
          case Ifcond:
            if (visible(node->y, node->height)) {
              if (strlen(node->data.text) > 1) {
                XDrawLine(gXDisplay, gWindow->fDisplayedWindow,
                  gWindow->fStandardGC, node->x,
                  node->y + gRegionOffset + yOff -
                    gTopOfGroupStack->cur_font->descent - word_off_height,
                  node->x + node->width,
                  node->y + gRegionOffset + yOff - word_off_height -
                    gTopOfGroupStack->cur_font->descent);
              }
              else {
               XDrawString(gXDisplay, gWindow->fDisplayedWindow,
                 gWindow->fStandardGC, node->x, node->y +
                   gRegionOffset - gTopOfGroupStack->cur_font->descent + yOff,
```

```
        node->data.text, 1);
    }
}
else {
  if (above(node->y))
    need_scroll_up_button = 1;
  else if (below(node->y))
    need_scroll_down_button = 1;
}
break;
case Lsquarebrace:
case Math:
case Punctuation:
case Rsquarebrace:
case Spadsrctxt:
case WindowId:
case Word:
  if (visible(node->y, node->height))
    XDrawString(gXDisplay, gWindow->fDisplayedWindow,
      gWindow->fStandardGC, node->x, node->y +
        gRegionOffset - gTopOfGroupStack->cur_font->descent + yOff,
      node->data.text, node->width);
  else {
    if (above(node->y))
      need_scroll_up_button = 1;
    else if (below(node->y))
      need_scroll_down_button = 1;
  }
  break;
case Verbatim:
  pushGroupStack();
  ttTopGroup();
  if (visible(node->y, node->height))
    XDrawString(gXDisplay, gWindow->fDisplayedWindow,
      gWindow->fStandardGC, node->x, node->y +
        gRegionOffset - gTopOfGroupStack->cur_font->descent + yOff,
      node->data.text, node->width);
  else {
    if (above(node->y))
      need_scroll_up_button = 1;
    else if (below(node->y))
      need_scroll_down_button = 1;
  }
  popGroupStack();
  break;
case Horizontalline:
```

```
    if (visible(node->y, node->height)) {
      lineTopGroup();
      XDrawLine(gXDisplay, gWindow->fDisplayedWindow,
        gWindow->fStandardGC, 0, node->y + gRegionOffset + yOff,
        gWindow->width, node->y + gRegionOffset + yOff);
      popGroupStack();
    }
    else {
      if (above(node->y))
        need_scroll_up_button = 1;
      else if (below(node->y))
        need_scroll_down_button = 1;
    }
    break;
  case Box:
    if (visible(node->y, node->height))
      XDrawRectangle(gXDisplay, gWindow->fDisplayedWindow,
        gWindow->fStandardGC, node->x,
        node->y + gRegionOffset + yOff - node->height,
        node->width, node->height);
    else {
      if (above(node->y))
        need_scroll_up_button = 1;
      else if (below(node->y))
        need_scroll_down_button = 1;
    }
    break;
  case Downlink:
  case Link:
  case LispDownLink:
  case LispMemoLink:
  case Lispcommand:
  case Lispcommandquit:
  case Lisplink:
  case Lispwindowlink:
  case Memolink:
  case Qspadcall:
  case Qspadcallquit:
  case Returnbutton:
  case Spadcall:
  case Spadcallquit:
  case Spaddownlink:
  case Spadlink:
  case Spadmemolink:
  case Unixcommand:
  case Unixlink:
```

```
case Upbutton:
case Windowlink:
  if (pix_visible(node->y, node->height))
      showLink(node);
  break;
case Spadcommand:
case Spadgraph:
case Spadsrc:
  showSpadcommand(node);
  break;
case Pastebutton:
  if (visible(node->y, node->height))
      showPastebutton(node);
  break;
case Paste:
  showPaste(node);
  break;
case Group:
case Tableitem:
  pushGroupStack();
  break;
case Controlbitmap:
  showImage(node, gWindow->fControlGC);
  break;
case Inputbitmap:
  showImage(node, gWindow->fStandardGC);
  break;
case Inputpixmap:
  showImage(node, gWindow->fStandardGC);
  break;
case BoldFace:
  bfTopGroup();
  break;
case Emphasize:
  if (gTopOfGroupStack->cur_font == gRmFont)
      emTopGroup();
  else
      rmTopGroup();
  break;
case It:
  emTopGroup();
  break;
case Sl:
case Rm:
  rmTopGroup();
  break;
```

```
case Tt:
  ttTopGroup();
  break;
case Inputstring:
  showInput(node);
  break;
case Radiobox:
case SimpleBox:
  showSimpleBox(node);
  break;
case Beep:
  LoudBeepAtTheUser();
  break;
case Description:
  bfTopGroup();
  break;
case Endspadsrc:
case Endspadcommand:
  gInAxiomCommand = 1;
case Endtableitem:
case Enddescription:
case Endpastebutton:
case Endlink:
case Endbutton:
case Endgroup:
  popGroupStack();
case Endverbatim:
case Endmath:
case Endbox:
case Endtable:
case Endmbox:
case Endparameter:
case Endpaste:
case Endinputbox:
case Endcenter:
case Endmacro:
case Endif:
case Endtitems:
case Enditems:
  /*
   * Now since I can show specific regions of the text, then at
   * this point I should check to see if I am the end
   */
  if (node->type == Ender)
      return;
  break;
```

```
            case Endfooter:
            case Endscrolling:
            case Endheader:
            case Endtitle:
              /*
               * regardless of what ender I have, I always terminate showing
               * with one of these
               */
              return;
            default:
              fprintf(stderr, "showText: Unknown Node Type %d\n", node->type);
              break;
          }
      }
  }
```

9.29.2 showLink

⟨*hypertex*⟩+≡

```
static void showLink(TextNode *node) {
    XWindowChanges wc;
    int active;
    switch (node->type) {
      case Upbutton:
        if (!need_up_button) {
            XClearArea(gXDisplay, gWindow->fDisplayedWindow, node->x,
                       node->y - node->height + gRegionOffset,
                       node->width, node->height, 0);
            active = 0;
        }
        else
            active = 1;
        break;
      case Returnbutton:
        if (!need_return_button) {
            XClearArea(gXDisplay, gWindow->fDisplayedWindow, node->x,
                       node->y - node->height + gRegionOffset,
                       node->width, node->height, 0);
            active = 0;
        }
        else
            active = 1;
        break;
      case Helpbutton:
        if (!need_help_button) {
            XClearArea(gXDisplay, gWindow->fDisplayedWindow, node->x,
                       node->y - node->height + gRegionOffset,
                       node->width, node->height, 0);
            active = 0;
        }
        else
            active = 1;
        break;
      default:
        active = 1;
        break;
    }
    if (active) {
        ButtonList *bl = allocButtonList();
        pushActiveGroup();
        wc.x = node->x;
        wc.y = node->y - node->height + yOff + gRegionOffset;
```

```
        wc.height = node->height;
        wc.width = node->width - trailingSpace(node->next);
        bl->x0 = wc.x;
        bl->y0 = wc.y;
        bl->x1 = bl->x0 + wc.width;
        bl->y1 = bl->y0 + wc.height;
        bl->link = node->link;
        if (!not_in_scroll) {
            bl->y0 += gWindow->page->top_scroll_margin + scroll_top_margin;
            bl->y1 += gWindow->page->top_scroll_margin + scroll_top_margin;
            bl->next = gWindow->page->s_button_list;
            gWindow->page->s_button_list = bl;
        }
        else {
            bl->next = gWindow->page->button_list;
            gWindow->page->button_list = bl;
        }
    }
    else
        rmTopGroup();
}
```

9.29.3 showPaste

⟨*hypertex*⟩+≡

```
static void showPaste(TextNode *node) {
    PasteNode *paste;
    if (!(paste = (PasteNode *) hashFind(gWindow->fPasteHashTable,
        node->data.text)))
            return;
    /*
     * Once I have got this far, then I had better save the current group
     * stack and the item stack
     */
    if (paste->group)
        freeGroupStack(paste->group);
    paste->group = (GroupItem *) copyGroupStack();
    if (paste->item_stack)
        freeItemStack(paste->item_stack);
    paste->item_stack = (ItemStack *) copyItemStack();
}
```

9.29.4 showPastebutton

⟨*hypertex*⟩+≡

```
static void showPastebutton(TextNode *node) {
    XWindowChanges wc;
    pushActiveGroup();
    wc.x = node->x;
    wc.y = node->y - node->height + yOff + gRegionOffset;
    wc.height = node->height;
    wc.width = node->width - trailingSpace(node->next);
#ifdef DEBUG
    fprintf(stderr, "Configure in  showLink %d %d %d %d\n",
            wc.x, wc.y, wc.width, wc.height);
#endif
    XConfigureWindow(gXDisplay, node->link->win,
        CWX | CWY | CWHeight | CWWidth, &wc);
    XMapWindow(gXDisplay, node->link->win);
}
```

9.29.5 showInput

Display an input string window.

⟨*hypertex*⟩+≡

```
static void showInput(TextNode *node) {
    XWindowChanges wc;
    InputItem *item;
    char *inpbuffer;
    item = node->link->reference.string;
    inpbuffer = item->curr_line->buffer;
    wc.border_width = 0;
    wc.x = node->x;
    wc.y = node->y + gRegionOffset + yOff - node->height + 2;
    wc.height = node->height - 2;
    wc.width = node->width;
    if (pix_visible(node->y, node->height)) {
        XConfigureWindow(gXDisplay, node->link->win,
                         CWX | CWY | CWHeight | CWWidth | CWBorderWidth,
                         &wc);
        XMapWindow(gXDisplay, node->link->win);
    }
    XFlush(gXDisplay);
    drawInputsymbol(item);
}
```

9.29.6 showSimpleBox

⟨*hypertex*⟩+≡

```
static void showSimpleBox(TextNode *node) {
    XWindowChanges wc;
    InputBox *box;
    /* first configure the box size properly */
    box = node->link->reference.box;
    wc.x = node->x;
    wc.y = node->y + gRegionOffset + yOff - node->height;
    wc.height = ((box->picked) ?
                 (box->selected->height) : (box->unselected->height));
    wc.width = node->width;
    if (visible(node->y + gTopOfGroupStack->cur_font->ascent, node->height)) {
      XConfigureWindow(gXDisplay, node->link->win,
        CWX | CWY | CWHeight | CWWidth, &wc);
        XMapWindow(gXDisplay, node->link->win);
        if (box->picked)
            pick_box(box);
        else
            unpick_box(box);
    }
}
```

9.29.7 showSpadcommand

Display a spad command node.

⟨*hypertex*⟩+≡

```
static void showSpadcommand(TextNode *node) {
    XWindowChanges wc;
    gInAxiomCommand = 1;
    pushSpadGroup();
    wc.x = node->x;
    if (node->type == Spadsrc)
        wc.y = node->y + gRegionOffset + yOff - 2 * node->height;
    else
        wc.y = node->y + gRegionOffset + yOff - node->height;
    wc.height = node->height;
    wc.width = node->width;
#ifdef DEBUG
    fprintf(stderr, "Spadcommand configured %d x %d -- (%d, %d)\n",
            wc.width, wc.height, wc.x, wc.y);
#endif
    XConfigureWindow(gXDisplay, node->link->win,
        CWX | CWY | CWHeight | CWWidth, &wc);
    XMapWindow(gXDisplay, node->link->win);
}
```

9.29.8 showImage

Display a pixmap.

⟨*hypertex*⟩+≡

```
static void showImage(TextNode *node, GC gc) {
    int src_x, src_y, src_width, src_height, dest_x, dest_y, ret_val;
    if (!pix_visible(node->y, node->height))
        return;
    if (node->image.xi == NULL)
        return;
    dest_x = node->x;
    src_x = 0;
    src_y = 0;
    dest_y = node->y + gRegionOffset - node->height + yOff;
    need_scroll_up_button = 1;
    if (node->width > (right_margin - node->x))
        src_width = right_margin - node->x;
    else
        src_width = node->width;

    if (gDisplayRegion != Scrolling) {
        src_y = 0;
        src_height = node->image.xi->height;
    }
    else {
        /* I may have only a partial image */
        if (dest_y < 0) {       /* the top is cut off */
            src_y = -dest_y;
            dest_y = 0;
            src_height = node->image.xi->height - src_y;
        }
        else if (dest_y + node->image.xi->height > gWindow->scrollheight) {
            /* the bottom is cut off */
            src_y = 0;
            src_height = gWindow->scrollheight - dest_y;
        }
        else {                  /* the whole thing is visible */
            src_y = 0;
            src_height = node->image.xi->height;
        }
    }
    ret_val = XPutImage(gXDisplay, gWindow->fDisplayedWindow, gc,
            node->image.xi, src_x, src_y, dest_x, dest_y,
            src_width, src_height);
    switch (ret_val) {
```

```
        case BadDrawable:
          fprintf(stderr, "(HyperDoc: showImage) bad drawable\n");
          break;
        case BadGC:
          fprintf(stderr, "(HyperDoc: showImage) bad GC");
          break;
        case BadMatch:
          fprintf(stderr, "(HyperDoc: showImage) bad match");
          break;
        case BadValue:
          fprintf(stderr, "(HyperDoc: showImage) bad value");
          break;
      }
  }
```

9.30 Axiom communication interface

Still a problem with closeClient.

9.30.1 issueSpadcommand

Issue a AXIOM command to the buffer associated with a page.

⟨*hypertex*⟩+≡

```
void issueSpadcommand(HyperDocPage *page, TextNode *command,
                      int immediate, int type) {
    char *buf;
    int ret_val;
    ret_val = connectSpad();
    if (ret_val == NotConnected || ret_val == SpadBusy)
        return;
    if (page->sock == NULL)
        startUserBuffer(page);
    ret_val = send_int(page->sock, TestLine);
    if (ret_val == -1) {
        page->sock = NULL;
        clearExecutionMarks(page->depend_hash);
        issueSpadcommand(page, command, immediate, type);
        return;
    }
    issueDependentCommands(page, command, type);
    ret_val = send_int(page->sock, ReceiveInputLine);
    buf = printToString(command);
    if (immediate) {
        buf[strlen(buf) + 1] = '\0';
        buf[strlen(buf)] = '\n';
    }
    if (type == Spadsrc)
        sendPile(page->sock, buf);
    else
        send_string(page->sock, buf);
    markAsExecuted(page, command, type);
    gIsEndOfOutput = 0;
}
```

9.30.2 sendPile

⟨*hypertex*⟩+≡

```
static void sendPile(Sock *sock,char * str) {
    FILE *f;
    char name[512], command[512];
    sprintf(name, "/tmp/hyper%s.input", getenv("SPADNUM"));
    f = fopen(name, "w");
    if (f == NULL) {
        fprintf(stderr, "Can't open temporary input file %s\n", name);
        return;
    }
    fprintf(f, "%s", str);
    fclose(f);
    sprintf(command, ")read %s\n", name);
    send_string(sock, command);
}
```

9.30.3 issueDependentCommands

⟨*hypertex*⟩+≡

```
static void issueDependentCommands(HyperDocPage *page,
                                   TextNode *command,int type) {
    TextNode *node, *depend_label;
    SpadcomDepend *depend;
    int endType = (type == Spadcommand || type == Spadgraph) ?
    (Endspadcommand) : (Endspadsrc);
    for (node = command->next; node->type != endType;
         node = node->next)
        if (node->type == Free)
            for (depend_label = node->data.node; depend_label != NULL;
                 depend_label = depend_label->next)
                if (depend_label->type == Word) {
                    depend = (SpadcomDepend *)
                        hashFind(page->depend_hash, depend_label->data.text);
                    if (depend == NULL) {
                        fprintf(stderr,
                                "Error: dependency on undefined label: %s\n",
                                depend_label->data.text);
                        continue;
                    }
                    if (!depend->executed) {
                        issueSpadcommand(page, depend->spadcom->next, 1,
                                         depend->spadcom->type);
                        while (!gIsEndOfOutput)
                            pause();
                        sleep(1);
                    }
                }
}
```

9.30.4 markAsExecuted

⟨*hypertex*⟩+≡

```
static void markAsExecuted(HyperDocPage *page, TextNode *command,int type) {
    TextNode *node, *depend_label;
    SpadcomDepend *depend;
    int endType = (type == Spadcommand || type == Spadgraph)
    ? (Endspadcommand) : (Endspadsrc);
    for (node = command; node->type != endType; node = node->next)
        if (node->type == Bound)
            for (depend_label = node->data.node; depend_label != NULL;
                 depend_label = depend_label->next)
                if (depend_label->type == Word) {
                    depend = (SpadcomDepend *)
                        hashFind(page->depend_hash, depend_label->data.text);
                    if (depend == NULL) {
                        fprintf(stderr, "No dependency entry for label: %s\n",
                                depend_label->data.text);
                        continue;
                    }
                    depend->executed = 1;
                }
}
```

9.30.5 startUserBuffer

Start a spad buffer for the page associated with the give.

⟨*hypertex*⟩+≡

```
static void startUserBuffer(HyperDocPage *page) {
    char buf[1024], *title;
    char *SPAD;
    char spadbuf[250];
    char complfile[250];
    int ret_val;
    SPAD = (char *) getenv("AXIOM");
    if (SPAD == NULL) {
        sprintf(SPAD, "/spad/mnt/rios");
    }
    sprintf(spadbuf, "%s/lib/spadbuf", SPAD);
    sprintf(complfile, "%s/lib/command.list", SPAD);
    title = printToString(page->title);
    if (access(complfile, R_OK) == 0)
        /*
         * TTT says : why not invoke with "-name axiomclient" and set any
         * defaults in the usual way
         */
#ifdef RIOSplatform
        sprintf(buf,
        "aixterm -sb -sl 500 -name axiomclient -n '%s' -T '%s'  -e  %s %s %s&",
        title, title, spadbuf, page->name, complfile);
    else
        sprintf(buf,
         "aixterm -sb -sl 500 -name axiomclient -n '%s' -T '%s' -e  %s %s&",
                title, title, spadbuf, page->name);
#else
#ifdef SUNplatform
        sprintf(buf,
        "xterm -sb -sl 500 -name axiomclient -n '%s' -T '%s' -e  %s %s %s&",
                title, title, spadbuf, page->name, complfile);
    else
        sprintf(buf,
           "xterm -sb -sl 500 -name axiomclient -n '%s' -T '%s' -e  %s %s&",
                title, title, spadbuf, page->name);
#else
        sprintf(buf,
        "xterm -sb -sl 500 -name axiomclient -n '%s' -T '%s' -e  %s %s %s&",
                title, title, spadbuf, page->name, complfile);
    else
        sprintf(buf,
```

```
          "xterm -sb -sl 500 -name axiomclient -n '%s' -T '%s' -e  %s '%s'&",
                 title, title, spadbuf, page->name);
#endif
#endif
    ret_val = system(buf);
    if (ret_val == -1 || ret_val == 127) {
        /*
         * perror("running the xterm spadbuf program"); exit(-1);
         */
    }
    acceptMenuServerConnection(page);
    sleep(2);
}
```

9.30.6 clearExecutionMarks

Clears the execution marks in a hash table when a buffer has been killed.

⟨*hypertex*⟩+≡

```
static void clearExecutionMarks(HashTable *depend_hash) {
    int i;
    HashEntry *h;
    SpadcomDepend *depend;
    if (depend_hash == 0)
        return;
    for (i = 0; i < depend_hash->size; i++)
        for (h = depend_hash->table[i]; h != NULL; h = h->next) {
            depend = (SpadcomDepend *) h->data;
            depend->executed = 0;
        }
}
```

9.30.7 acceptMenuConnection

⟨*hypertex*⟩+≡

```
Sock *acceptMenuConnection(Sock *server_sock) {
    int sock_fd;
    Sock_List *pls;
    /* Could only be InterpWindow */
    pls = (Sock_List *) halloc(sizeof(Sock_List),"SockList");
    sock_fd = accept(server_sock->socket, 0, 0);
    if (sock_fd == -1) {
        perror("session : accepting connection");
        return 0;
    }
    (pls->Socket).socket = sock_fd;
    get_socket_type((Sock *) pls);
#ifdef DEBUG
    fprintf(stderr,
            "session: accepted InterpWindow , fd = %d\n", sock_fd);
#endif
    /* put new item at head of list */
    if (plSock == (Sock_List *) 0) {
        plSock = pls;
        plSock->next = (Sock_List *) 0;
    }
    else {
        pls->next = plSock;
        plSock = pls;
    }
    /* need to maintain socket_mask since we roll our own accept */
    FD_SET(plSock->Socket.socket, &socket_mask);
    return (Sock *) plSock;
}
```

9.30.8 acceptMenuServerConnection

TTT thinks this code should just provide a Sock to the page. The only client assumed is a spadbuf. Since spadbuf was invoked with the page name, it just passes it back here as a check (`get_string` line).

⟨*hypertex*⟩+≡

```
static void acceptMenuServerConnection(HyperDocPage *page) {
    int ret_code/*, i*/;
    fd_set rd;
    Sock *sock;
    char *buf_name;
    HyperDocPage *npage;
    if (page->sock != NULL)
        return;
    while (1) {
        rd = server_mask;
        ret_code = sselect(FD_SETSIZE, &rd, 0, 0, NULL);
        if (ret_code == -1) {
            perror("Session manager select");
            continue;
        }
        if (server[1].socket > 0 && FD_ISSET(server[1].socket, &rd)) {
            sock = acceptMenuConnection(server + 1);
            if (sock == 0)
                return;
            if (sock->purpose == InterpWindow) {
                buf_name = get_string(sock);
                npage = (HyperDocPage *)
                    hashFind(gWindow->fPageHashTable, buf_name);
                if (npage == NULL) {
                    /*
                     * Lets just try using the current page TTT doesn't know
                     * why this could be detrimental
                     *
                     * fprintf(stderr, "connecting spadbuf to unknown page:
                     * %s\n", buf_name);
                     */
                    page->sock = sock;
                    return;
                }
                else {
                    /*
                     * For some reason npage and page may be different TTT
                     * thinks this happens when a dynamic page has the same
                     * name as an existing static page.
```

```
                     */
                    npage->sock = sock;
                    page->sock = sock;
                }
                if (!strcmp(buf_name, page->name)) {
                    return;
                }
            }
        }
    }
}
```

9.30.9 printToString

This routine takes a text node and creates a string out of it. This is for use with things such as spad commands. There are a very limited set of node types it can handle, so be careful.

⟨*hypertex*⟩+≡

```
char *printToString(TextNode *command) {
    int len = 0;
    printToString1(command, &len);
    p2sBuf = resizeBuffer(len, p2sBuf, &p2sBufSize);
    return printToString1(command, NULL);
}
```

9.30.10 printToString1

⟨*hypertex*⟩+≡

```
char *printToString1(TextNode *command,int * sizeBuf) {
    char *c = p2sBuf;
    char *s;
    InputItem *item;
    LineStruct *curr_line;
    int lcount;
    InputBox *box;
    int num_spaces;
    int count;
    TextNode *node;
    /*
     * Init the stack of text nodes, things are pushed on here when I trace
     * through a nodes data.node. This way I always no where my next is.
     */
    for (node = command; node != NULL;) {
        switch (node->type) {
          case Newline:
            storeChar('\n');
            node = node->next;
            break;
          case Ifcond:
            if (checkCondition(node->data.ifnode->cond))
                node = node->data.ifnode->thennode;
            else
                node = node->data.ifnode->elsenode;
            break;
          case Endarg:
          case Endspadcommand:
          case Endspadsrc:
          case Endpix:
            storeChar('\0');
            return p2sBuf;
          case Endverbatim:
          case Endif:
          case Fi:
          case Endmacro:
          case Endparameter:
          case Rbrace:
          case Endgroup:
            node = node->next;
            break;
          case Punctuation:
            /*
```

```
     * Simply copy the piece of text
     */
    if (node->space & FRONTSPACE) { storeChar(' '); }
    for (s = node->data.text; *s; s++) { storeChar(*s); }
    node = node->next;
    break;
  case WindowId:
    /*
     * Simply copy the piece of text
     */
    if (node->space) { storeChar(' '); }
    for (s = node->data.text; *s; s++) { storeChar(*s); }
    storeChar(' ');
    node = node->next;
    break;
  case Verbatim:
  case Spadsrctxt:
    /*
     * Simply copy the piece of text
     */
    if (node->space) { storeChar(' '); }
    for (s = node->data.text; *s; s++) { storeChar(*s); }
    /*
     * now add the eol
     */
    /*
     * if(node->next && node->next->type != Endspadsrc)
     * storeChar('\n');
     */
    node = node->next;
    break;
  case Dash:
  case Rsquarebrace:
  case Lsquarebrace:
  case Word:
    /*
     * Simply copy the piece of text
     */
    if (node->space) { storeChar(' '); }
    for (s = node->data.text; *s; s++) { storeChar(*s); }
    node = node->next;
    break;
  case BoxValue:
    box =
     (InputBox *) hashFind(gWindow->page->box_hash, node->data.text);
    if (box == NULL) {
```

```
            fprintf(stderr,
                    "printToString:Box %s Has no symbol table entry\n",
                    node->data.text);
            exit(-1);
        }
        storeChar(' ');
        if (box->picked) {
            storeChar('t');
        }
        else {
            storeChar('n');
            storeChar('i');
            storeChar('l');
        }
        node = node->next;
        break;
    case StringValue:
        item = returnItem(node->data.text);
        if (item != NULL) {
            if (node->space) { storeChar(' '); }
            curr_line = item->lines;
            while (curr_line != NULL) {
                for (lcount = 0,
                     s = curr_line->buffer; *s && lcount < item->size;
                     s++, lcount++) {
                  storeChar(funnyUnescape(*s));
                }
                if (curr_line->len <= item->size && curr_line->next) {
                    storeChar('\n');
                }
                curr_line = curr_line->next;
            }
        }
        else if ((box = (InputBox *) hashFind(gWindow->page->box_hash,
                                              node->data.text)) != NULL) {
            if (node->space) { storeChar(' '); }
            if (box->picked) {
                storeChar('t');
            }
            else {
                storeChar('n');
                storeChar('i');
                storeChar('l');
            }
        }
        else {
```

```
        fprintf(stderr, "Error, Symbol %s has no symbol table entry\n",
                node->data.text);
        exit(-1);
    }
    node = node->next;
    break;
  case Space:
    num_spaces = (node->data.node != NULL ?
                  atoi(node->data.node->data.text) : 1);
    for (count = 0; count < num_spaces; count++)
        storeChar(' ');
    node = node->next;
    break;
  case Titlenode:
  case Endtitle:
  case Center:
  case Endcenter:
  case BoldFace:
  case Emphasize:
  case Indentrel:
    node = node->next;
    break;
  case Bound:
    if (include_bf) {
        int len, i;
        TextNode *n2 = node->data.node;
        storeChar('\\');
        storeChar('b');
        storeChar('o');
        storeChar('u');
        storeChar('n');
        storeChar('d');
        storeChar('{');
        for (; n2->type != Endarg; n2 = n2->next) {
            if (n2->type == Word) {
                len = strlen(n2->data.text);
                for (i = 0; i < len; i++)
                    storeChar(n2->data.text[i]);
                storeChar(' ');
            }
        }
        storeChar('}');
    }
    node = node->next;
    break;
  case Free:
```

```
            if (include_bf) {
                int len, i;
                TextNode *n2 = node->data.node;
                storeChar('\\');
                storeChar('f');
                storeChar('r');
                storeChar('e');
                storeChar('e');
                storeChar('{');
                for (; n2->type != Endarg; n2 = n2->next) {
                    if (n2->type == Word) {
                        len = strlen(n2->data.text);
                        for (i = 0; i < len; i++)
                            storeChar(n2->data.text[i]);
                        storeChar(' ');
                    }
                }
                storeChar('}');
            }
            node = node->next;
            break;
        case Macro:
            node = node->next;
            break;
        case Pound:
            if (node->space) { storeChar(' '); }
            node = node->next;
            break;
        case Group:
            node = node->next;
            break;
        case Indent:
            num_spaces = (node->data.node != NULL ?
                          atoi(node->data.node->data.text) : 1);
            for (count = 0; count < num_spaces; count++)
                storeChar(' ');
            node = node->next;
            break;
        default:
            fprintf(stderr,
                    "printToString: Unrecognized Keyword Type %d\n",
                    node->type);
            node=node->next;
            break;
    }
}
```

```
    storeChar('\0');
    return p2sBuf;
}

/*
 * Send a lisp or spad command to the AXIOM server for execution , if
 * type is link, then we wait for a HyperDoc card to be returned
 */
```

9.30.11 issueServerCommand

⟨*hypertex*⟩+≡

```
HyperDocPage *issueServerCommand(HyperLink *link) {
    TextNode *command = (TextNode *) link->reference.node;
    int ret_val;
    char *buf;
    HyperDocPage *page;
    ret_val = connectSpad();
    if (ret_val == NotConnected) {
      page = (HyperDocPage *) hashFind(gWindow->fPageHashTable,
             "SpadNotConnectedPage");
      if (page == NULL)
            fprintf(stderr, "No SpadNotConnectedPage found\n");
      return page;
    }
    if (ret_val == SpadBusy) {
        page = (HyperDocPage *) hashFind(gWindow->fPageHashTable,
               "SpadBusyPage");
        if (page == NULL)
            fprintf(stderr, "No SpadBusyPage found\n");
        return page;
    }
    switchFrames();
    switch (link->type) {
      case Qspadcall:
      case Qspadcallquit:
      case Spadlink:
      case Spaddownlink:
      case Spadmemolink:
        send_int(spadSocket, QuietSpadCommand);
        break;
      case Spadcall:
      case Spadcallquit:
        send_int(spadSocket, SpadCommand);
        break;
      default:
        send_int(spadSocket, LispCommand);
        break;
    }
    buf = printToString(command);
    send_string(spadSocket, buf);
    if (link->type == Lispcommand || link->type == Spadcall
        || link->type == Spadcallquit || link->type == Qspadcallquit
        || link->type == Qspadcall || link->type == Lispcommandquit)
        return NULL;
```

```
    page = parsePageFromSocket();
    return page;
}
```

9.30.12 issueServerpaste

⟨*hypertex*⟩+≡

```
int issueServerpaste(TextNode *command) {
    char *buf;
    int ret_val;
    ret_val = connectSpad();
    if (ret_val == NotConnected || ret_val == SpadBusy)
        return 1;
    switchFrames();
    send_int(spadSocket, LispCommand);
    buf = printToString(command);
    send_string(spadSocket, buf);
    return 1;
}
```

9.30.13 issueUnixcommand

⟨*hypertex*⟩+≡

```
void issueUnixcommand(TextNode *node) {
    char *buf;
    char *copy;
    buf = printToString(node);
    copy =(char *) halloc((strlen(buf)+2)*sizeof(char),"Unixcommand");
    strcpy(copy,buf);
    copy[strlen(buf) + 1] = '\0';
    copy[strlen(buf)] = '&';
    system(copy);
    free(copy);
    return;
}
```

9.30.14 issueUnixlink

⟨*hypertex*⟩+≡

```
HyperDocPage *issueUnixlink(TextNode *node) {
    HyperDocPage *page;
    char *buf;
    buf = printToString(node);
    if ((unixfd = popen(buf, "r")) == NULL) {
        fprintf(stderr, "Error popening %s\n", buf);
        exit(-1);
    }
    bsdSignal(SIGUSR2,SIG_IGN,0);
    page = parsePageFromUnixfd();
    bsdSignal(SIGUSR2,sigusr2Handler,0);
    return page;
}
```

9.30.15 issueUnixpaste

⟨*hypertex*⟩+≡

```
int issueUnixpaste(TextNode *node) {
    char *buf;
    buf = printToString(node);
    if ((unixfd = popen(buf, "r")) == NULL) {
        fprintf(stderr, "Error popening %s\n", buf);
        exit(-1);
    }
    return 1;
}
```

9.30.16 serviceSessionSocket

Called when sessionServer selects.

⟨*hypertex*⟩+≡

```
void serviceSessionSocket(void) {
    int cmd, pid;
    cmd = get_int(sessionServer);
    switch (cmd) {
      case CloseClient:
        pid = get_int(sessionServer);
        if (pid != -1)
            closeClient(pid);
        break;
      default:
        fprintf(stderr,
                "(HyperDoc) Unknown command from SessionServer %d\n", cmd);
        break;
    }
}
```

9.30.17 switchFrames

Let spad know which frame to issue command via

⟨*hypertex*⟩+≡

```
static void switchFrames(void) {
    if (sessionServer == NULL) {
        fprintf(stderr, "(HyperDoc) No session manager connected!\n");
        return;
    }
    if (gWindow->fAxiomFrame == -1) {
        fprintf(stderr,
             "(HyperDoc) No AXIOM frame associated with top level window!\n");
        return;
    }
    send_int(sessionServer, SwitchFrames);
    send_int(sessionServer, gWindow->fAxiomFrame);
}
```

9.30.18 sendLispCommand

⟨*hypertex*⟩+≡

```
void sendLispCommand(char *command) {
    int ret_val;
    ret_val = connectSpad();
    if (ret_val == NotConnected || ret_val == SpadBusy) {
        return;
    }
    send_int(spadSocket, LispCommand);
    send_string(spadSocket, command);
}
```

9.30.19 escapeString

⟨*hypertex*⟩+≡

```
void escapeString(char *s) {
    char *st;
    for (st = s; *st; st++)
        *st = funnyEscape(*st);
}
```

9.30.20 unescapeString

⟨*hypertex*⟩+≡

```
void unescapeString(char *s) {
    char *st;
    for (st = s; *st; st++)
        *st = funnyUnescape(*st);
}
```

9.30.21 closeClient

⟨*hypertex*⟩+≡

```
static void closeClient(int pid) {
    Sock_List *pSock, *locSock;
    /*
     * just need to drop the list item
     */
    if (plSock == (Sock_List *) 0)
        return;
    /*
     * first check head
     */
    if ((plSock->Socket.pid == pid)) {
        locSock = plSock;
        if ((*plSock).next == (Sock_List *) 0) {
            plSock = (Sock_List *) 0;
        }
        else {
            plSock = plSock->next;
        }
        free(locSock);
    }
    /*
     * now check the rest
     */
    else {
        for (pSock = plSock;
             pSock->next != (Sock_List *) 0;
             pSock = pSock->next)
          if (pSock->next->Socket.pid == pid) {
                locSock = pSock->next;
                if (pSock->next->next == (Sock_List *) 0) {
                    pSock->next = (Sock_List *) 0;
                }
                else {
                    pSock->next = pSock->next->next;
                }
                free(locSock);
                break;
          }
    }
}
```

9.30.22 printSourceToString

⟨*hypertex*⟩+≡

```
char *printSourceToString(TextNode *command) {
    int len = 0;
    printSourceToString1(command, &len);
    p2sBuf = resizeBuffer(len, p2sBuf, &p2sBufSize);
    return printSourceToString1(command, NULL);
}
```

9.30.23 printSourceToString1

⟨*hypertex*⟩+≡

```
char *printSourceToString1(TextNode *command,int * sizeBuf) {
    char *c = p2sBuf;
    char *s;
    InputItem *item;
    LineStruct *curr_line;
    int lcount;
    InputBox *box;
    int num_spaces;
    int count;
    TextNode *node;
    /* print out HyperDoc source for what you see */
    for (node = command; node != NULL;) {
        switch (node->type) {
          case Newline:
            storeString("\\newline\n");
            node = node->next;
            break;
          case Par:
            storeString("\n\n");
            node = node->next;
            break;
          case Indentrel:
            storeString("\\indentrel{");
            storeString(node->data.node->data.text);
            storeChar('}');
            node = node->next;
            break;
          case Tab:
            storeString("\\tab{");
            storeString(node->data.node->data.text);
            storeChar('}');
            node = node->next;
            break;
          case Ifcond:
            if (checkCondition(node->data.ifnode->cond))
                node = node->data.ifnode->thennode;
            else
                node = node->data.ifnode->elsenode;
            break;
          case Endarg:
          case Endspadsrc:
          case Endpix:
          case Endbutton:
```

```
  storeChar('}');
  node = node->next;
  break;
case Endverbatim:
case Endif:
case Fi:
case Endmacro:
case Endparameter:
case Rbrace:
  node = node->next;
  break;
case Punctuation:
  /*
   * Simply copy the piece of text
   */
  if (node->space & FRONTSPACE) { storeChar(' '); }
  for (s = node->data.text; *s; s++) { storeChar(*s); }
  node = node->next;
  break;
case WindowId:
  storeString("\\windowid ");
  node = node->next;
  break;
case Verbatim:
case Spadsrctxt:
  if (node->space) { storeChar(' '); }
  for (s = node->data.text; *s; s++) { storeChar(*s); }
  node = node->next;
  break;
case Dash:
case Rsquarebrace:
case Lsquarebrace:
case Word:
  if (node->space) { storeChar(' '); }
  for (s = node->data.text; *s; s++) { storeChar(*s); }
  node = node->next;
  break;
case BoxValue:
  box=(InputBox *)hashFind(gWindow->page->box_hash,node->data.text);
  if (box == NULL) {
      fprintf(stderr,
         "printToString:Box %s Has no symbol table entry\n",
         node->data.text);
      exit(-1);
  }
  storeChar(' ');
```

```
            if (box->picked) {
                storeChar('t');
            }
            else {
                storeChar('n');
                storeChar('i');
                storeChar('l');
            }
            node = node->next;
            break;
          case StringValue:
            item = returnItem(node->data.text);
            if (item != NULL) {
                if (node->space) {  storeChar(' '); }
                curr_line = item->lines;
                while (curr_line != NULL) {
                    for (lcount = 0, s = curr_line->buffer;
                         *s && lcount < item->size;
                         s++, lcount++) {
                        storeChar(funnyUnescape(*s));
                    }
                    if (curr_line->len <= item->size && curr_line->next) {
                        storeChar('\n');
                    }
                    curr_line = curr_line->next;
                }
            }
            else if ((box = (InputBox *) hashFind(gWindow->page->box_hash,
                                                  node->data.text)) != NULL) {
                if (node->space) { storeChar(' '); }
                if (box->picked) {
                    storeChar('t');
                }
                else {
                    storeChar('n');
                    storeChar('i');
                    storeChar('l');
                }
            }
            else {
                fprintf(stderr, "Error, Symbol %s has no symbol table entry\n",
                        node->data.text);
                exit(-1);
            }
            node = node->next;
            break;
```

```
            case Space:
              num_spaces = (node->data.node != NULL ?
                            atoi(node->data.node->data.text) : 1);
              for (count = 0; count < num_spaces; count++)
                  storeChar(' ');
              node = node->next;
              break;
            case Emphasize:
              storeString("\\em ");
              node = node->next;
              break;
            case BoldFace:
              storeString("\\bf ");
              node = node->next;
              break;
            case Sl:
              storeString("\\it ");
              node = node->next;
              break;
            case Rm:
              storeString("\\rm ");
              node = node->next;
              break;
            case It:
              storeString("\\it ");
              node = node->next;
              break;
            case Tt:
              storeString("\\tt ");
              node = node->next;
              break;
            case Group:
 /* skip {} */
              if (node->next->type==Endgroup){
                 node=node->next->next;
                 break;
              }
              storeChar('{');
              node = node->next;
              break;
            case Endgroup:
              storeChar('}');
              node = node->next;
              break;
            case Box:
              storeString("\\box{");
```

```
  node = node->next;
  break;
case Endbox:
  storeChar('}');
  node = node->next;
  break;
case Center:
  storeString("\\center{");
  node = node->next;
  break;
case Endcenter:
  storeString("}");
  storeChar('\n');
  node = node->next;
  break;
case Titlenode:
case Endtitle:
  node = node->next;
  break;
case Bound:
  {
      TextNode *n2 = node->data.node;
      storeString("\\bound{");
      for (; n2->type != Endarg; n2 = n2->next) {
          if (n2->type == Word) {
              storeString(n2->data.text);
              storeChar(' ');
              }
          }
      storeChar('}');
  }
  node = node->next;
  break;
case Free:
  {
      TextNode *n2 = node->data.node;
      storeString("\\free{");
      for (; n2->type != Endarg; n2 = n2->next) {
          if (n2->type == Word) {
              storeString(n2->data.text);
              storeChar(' ');
              }
          }
      storeChar('}');
      }
  node = node->next;
```

```
  break;
case Macro:
  storeChar(' ');
  node = node->next;
  break;
case Pound:
  if (node->space) { storeChar(' '); }
  node = node->next;
  break;
case Indent:
  num_spaces = (node->data.node != NULL ?
                atoi(node->data.node->data.text) : 1);
  for (count = 0; count < num_spaces; count++)
      storeChar(' ');
  node = node->next;
  break;
case Inputbitmap:
  storeString("\\inputbitmap{");
  storeString(node->data.text);
  storeString("}\n");
  node = node->next;
  break;
case Endscrolling:
  storeString("\\end{scroll}\n");
  node = node->next;
  break;
case Scrollingnode:
  storeString("\\begin{scroll}\n");
  storeString("% This is the scrolling area\n");
  node = node->next;
  break;
case Horizontalline:
  storeString("\\horizontalline\n");
  node = node->next;
  break;
case Endtable:
  storeChar('}');
  node = node->next;
  break;
case Table:
  storeString("\\table{");
  node = node->next;
  break;
case Tableitem:
  storeChar('{');
  node = node->next;
```

```
            break;
          case Endtableitem:
            storeChar('}');
            node = node->next;
            break;
          case Beginitems:
            storeString("\\begin{items}");
            node = node->next;
            break;
          case Item:
            storeString("\n\\item");
            node = node->next;
            break;
          case Enditems:
            storeString("\n\\end{items}");
            node = node->next;
            break;
/*** LINKS ***/
/* all these guys are ended by Endbutton
we close the brace then */
          case Spadlink:
            storeString("\\fauxspadlink{");
            node = node->next;
            break;
          case Unixlink:
            storeString("\\fauxunixlink{");
            node = node->next;
            break;
          case Lisplink:
            storeString("\\fauxlisplink{");
            node = node->next;
            break;
          case Link:
            storeString("\\fauxlink{");
            node = node->next;
            break;
          case LispDownLink:
            storeString("\\fauxlispdownlink{");
            node = node->next;
            break;
          case LispMemoLink:
            storeString("\\fauxlispmemolink{");
            node = node->next;
            break;
          case Memolink:
            storeString("\\fauxmemolink{");
```

```
            node = node->next;
            break;
          case Windowlink:
            storeString("\\fauxwindowlink{");
            node = node->next;
            break;
          case Downlink:
            storeString("\\fauxdownlink{");
            node = node->next;
            break;
/** END OF LINKS **/
          case Unixcommand:
            storeString("\\unixcommand{");
            node = node->next;
            break;
          case Lispcommand:
            storeString("\\lispcommand{");
            node = node->next;
            break;
          case Spadgraph:
            storeString("\\spadgraph{");
            node = node->next;
            break;
          case Spadcommand:
            storeString("\\spadcommand{");
            node = node->next;
            break;
          case Endspadcommand:
            storeChar('}');
            node = node->next;
            break;
          case Footernode:
            storeString("% This is the footer\n");
            node = node->next;
            break;
          case Endfooter:
            storeString("% This is the end of the footer\n");
            node = node->next;
            break;
          case Endheader:
            storeString("% This is the end of the header\n");
            node = node->next;
            break;
          case Headernode:
            storeString("% This is the header\n");
            node = node->next;
```

```
            break;
          default:
            fprintf(stderr,
                    "printToString: Unrecognized Keyword Type %d\n",
                    node->type);
            node=node->next;
            break;
        }
    }
    storeChar('\0');
    return p2sBuf;
}
```

9.31 Produce titlebar

9.31.1 makeTitleBarWindows

⟨*hypertex*⟩+≡

```
void makeTitleBarWindows(void) {
    XSetWindowAttributes at;
    unsigned long valuemask = 0L;
    /* read the images if we don't have them already */
    if (tw1image == NULL)
        readTitleBarImages();
    /* set the window attributes */
    at.cursor = gActiveCursor;
    valuemask |= CWCursor;
    at.event_mask = ButtonPress;
    valuemask |= CWEventMask;
    /* create the windows for the buttons */
    gWindow->fTitleBarButton1 =
      XCreateSimpleWindow(gXDisplay, gWindow->fMainWindow, 1, 1, twwidth,
                          twheight, 0, gBorderColor, BACKCOLOR);
    XChangeWindowAttributes(gXDisplay,gWindow->fTitleBarButton1,valuemask,&at);
    gWindow->fTitleBarButton2 =
      XCreateSimpleWindow(gXDisplay, gWindow->fMainWindow, 1, 1, twwidth,
                          twheight, 0, gBorderColor, BACKCOLOR);
    XChangeWindowAttributes(gXDisplay,gWindow->fTitleBarButton2,valuemask,&at);
    gWindow->fTitleBarButton3 =
      XCreateSimpleWindow(gXDisplay, gWindow->fMainWindow, 1, 1, twwidth,
        twheight, 0, gBorderColor, BACKCOLOR);
    XChangeWindowAttributes(gXDisplay,gWindow->fTitleBarButton3,valuemask,&at);
    gWindow->fTitleBarButton4 =
      XCreateSimpleWindow(gXDisplay, gWindow->fMainWindow, 1, 1, twwidth,
        twheight, 0, gBorderColor, BACKCOLOR);
    XChangeWindowAttributes(gXDisplay,gWindow->fTitleBarButton4,valuemask,&at);
}
```

9.31.2 showTitleBar

⟨*hypertex*⟩+≡

```
void showTitleBar(void) {
    XWindowChanges wc;
    int height, hbw = (int) gWindow->border_width / 2;
    XImage *image;
    /*
     * the first thing we do is pop up all the windows and
     * place them properly
     */
    if (gWindow->page->title->height != twheight)
        height = gWindow->page->title->height;
    else
        height = twheight;
    pushActiveGroup();
    /* configure and map button number 1 */
    wc.x = 0;
    wc.y = 0;
    wc.height = twheight;
    wc.width = twwidth;
    XConfigureWindow(gXDisplay, gWindow->fTitleBarButton1,
                     CWX | CWY | CWHeight | CWWidth, &wc);
    XMapWindow(gXDisplay, gWindow->fTitleBarButton1);
    image = tw1image;
    XPutImage(gXDisplay, gWindow->fTitleBarButton1, gWindow->BUTTGC,
              image, 0, 0, 0, 0, image->width, image->height);
    /* configure and map button number 2 */
    wc.x += twwidth + gWindow->border_width;
    XConfigureWindow(gXDisplay, gWindow->fTitleBarButton2,
                     CWX | CWY | CWHeight | CWWidth, &wc);
    XMapWindow(gXDisplay, gWindow->fTitleBarButton2);
    image = need_help_button ? tw2image : noopimage;
    XPutImage(gXDisplay, gWindow->fTitleBarButton2, gWindow->BUTTGC,
              image, 0, 0, 0, 0, image->width, image->height);
    /* configure and map button number 4 */
    wc.x = gWindow->width - twwidth;
    XConfigureWindow(gXDisplay, gWindow->fTitleBarButton4,
                     CWX | CWY | CWHeight | CWWidth, &wc);
    XMapWindow(gXDisplay, gWindow->fTitleBarButton4);
    image = need_up_button ? tw4image : noopimage;
    XPutImage(gXDisplay, gWindow->fTitleBarButton4, gWindow->BUTTGC,
              image, 0, 0, 0, 0, image->width, image->height);
    /* configure and map button number 3 */
    wc.x = wc.x - twwidth - gWindow->border_width;
    XConfigureWindow(gXDisplay, gWindow->fTitleBarButton3,
```

```
                    CWX | CWY | CWHeight | CWWidth, &wc);
    XMapWindow(gXDisplay, gWindow->fTitleBarButton3);
    image = need_return_button ? tw3image : noopimage;
    XPutImage(gXDisplay, gWindow->fTitleBarButton3, gWindow->BUTTGC,
              image, 0, 0, 0, 0, image->width, image->height);
    gWindow->fDisplayedWindow = gWindow->fMainWindow;
    gDisplayRegion = Title;
    gRegionOffset = 0;
    yOff = 0;
    popGroupStack();
    showText(gWindow->page->title->next, Endheader);
    /* Now draw the box around the title */
    lineTopGroup();
    XDrawLine(gXDisplay, gWindow->fMainWindow, gWindow->fStandardGC, 0,
              height + hbw, gWindow->width, height + hbw);
    popGroupStack();
}
```

9.31.3 linkTitleBarWindows

⟨*hypertex*⟩+≡

```
void linkTitleBarWindows(void) {
    HyperLink *tw1link = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink"),
              *tw2link = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink"),
              *tw3link = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink"),
              *tw4link = (HyperLink *) halloc(sizeof(HyperLink), "HyperLink");
    tw1link->win = gWindow->fTitleBarButton1;
    tw1link->type = Quitbutton;
    tw1link->reference.node = NULL;
    tw1link->x = tw1link->y = 0;
    tw2link->win = gWindow->fTitleBarButton2;
    tw2link->type = Helpbutton;
    tw2link->reference.node = NULL;
    tw2link->x = tw2link->y = 0;
    tw3link->win = gWindow->fTitleBarButton3;
    tw3link->type = Returnbutton;
    tw3link->reference.node = NULL;
    tw3link->x = tw3link->y = 0;
    tw4link->win = gWindow->fTitleBarButton4;
    tw4link->type = Upbutton;
    tw4link->reference.node = NULL;
    tw4link->x = tw4link->y = 0;
    hashInsert(gLinkHashTable, (char *)tw1link,(char *) &tw1link->win);
    hashInsert(gLinkHashTable, (char *)tw2link,(char *) &tw2link->win);
    hashInsert(gLinkHashTable, (char *)tw3link,(char *) &tw3link->win);
    hashInsert(gLinkHashTable, (char *)tw4link,(char *) &tw4link->win);
}
```

9.31.4 readTitleBarImages

⟨*hypertex*⟩+≡

```
static void readTitleBarImages(void) {
    int w, h;
    char filename[128];
    char *axiomEnvVar = NULL;
    axiomEnvVar = getenv("AXIOM");
    if (axiomEnvVar)
        sprintf(filename, "%s/doc/bitmaps/%s", axiomEnvVar, tw1file);
    else
        sprintf(filename, "%s", tw1file);
    tw1image = HTReadBitmapFile(gXDisplay, gXScreenNumber, filename,
                                &twwidth, &twheight);
    if (axiomEnvVar)
        sprintf(filename, "%s/doc/bitmaps/%s", axiomEnvVar, tw2file);
    else
        sprintf(filename, "%s", tw2file);
    tw2image = HTReadBitmapFile(gXDisplay, gXScreenNumber, filename, &w, &h);
    twwidth = ((twwidth >= w) ? (twwidth) : (w));
    if (axiomEnvVar)
        sprintf(filename, "%s/doc/bitmaps/%s", axiomEnvVar, tw3file);
    else
        sprintf(filename, "%s", tw3file);
    tw3image = HTReadBitmapFile(gXDisplay, gXScreenNumber, filename, &w, &h);
    twwidth = ((twwidth >= w) ? (twwidth) : (w));
    if (axiomEnvVar)
        sprintf(filename, "%s/doc/bitmaps/%s", axiomEnvVar, tw4file);
    else
        sprintf(filename, "%s", tw4file);
    tw4image = HTReadBitmapFile(gXDisplay, gXScreenNumber, filename, &w, &h);
    twwidth = ((twwidth >= w) ? (twwidth) : (w));
    if (axiomEnvVar)
        sprintf(filename, "%s/doc/bitmaps/%s", axiomEnvVar, noopfile);
    else
        sprintf(filename, "%s", noopfile);
    noopimage = HTReadBitmapFile(gXDisplay, gXScreenNumber, filename,
                                 &twwidth, &twheight);
}
```

9.31.5 getTitleBarMinimumSize

⟨*hypertex*⟩+≡

```
void getTitleBarMinimumSize(int *width, int *height) {
    (*width)  = 4 * twwidth + 40;
    (*height) = twheight + 2;
}
```

9.31.6 main

Initialize hash tables, signal handlers and windows, then call the main event handling loop

⟨*hypertex*⟩+≡

```
int main(int argc, char **argv) {
    int ret_status;
    /* Initialize some global values */
/*    fprintf(stderr,"hyper:main:entered\n");*/
    gArgc = argc;
    gArgv = argv;
    gIsEndOfOutput = 1;
/*    fprintf(stderr,"hyper:main:calling  checkArguments\n");*/
    checkArguments();
/*    fprintf(stderr,"hyper:main:returned checkArguments\n");*/
    /*
     * initialize the hash tables for the files and the windows and images
     */
/*    fprintf(stderr,"hyper:main:calling  initHash\n");*/
    initHash();
/*    fprintf(stderr,"hyper:main:returned initHash\n");*/
    /*
     * initialize the parser keyword hash table
     */
/*    fprintf(stderr,"hyper:main:calling  parserInit\n");*/
    parserInit();
/*    fprintf(stderr,"hyper:main:returned parserInit\n");*/
/*    fprintf(stderr,"hyper:main:calling  readHtDb\n");*/
    readHtDb(&init_page_hash, &init_macro_hash, &init_patch_hash);
/*    fprintf(stderr,"hyper:main:returned readHtDb\n");*/
    /*
     * Now initialize x. This includes opening the display, setting the
     * screen and display global values, and also gets all the fonts and
     * colors we will need.
     */
    if (!make_input_file && !gmakeRecord_file && !gverifyRecord_file) {
/*        fprintf(stderr,"hyper:main:calling  initializeWindowSystem\n");*/
        initializeWindowSystem();
/*        fprintf(stderr,"hyper:main:returned initializeWindowSystem\n");*/
        /*
         * Initialize some of the global values used by the input string
         * routines
         */
/*        fprintf(stderr,"hyper:main:calling  initKeyin\n");*/
        initKeyin();
```

```
/*        fprintf(stderr,"hyper:main:returned initKeyin\n");*/
        /*
         * regardless of what else happened, we should always pop up an
         * initial window.
         */
/*        fprintf(stderr,"hyper:main:calling  initTopWindow\n");*/
        ret_status = initTopWindow("RootPage");
/*        fprintf(stderr,"hyper:main:returned initTopWindow\n");*/
        gParentWindow = gWindow;
        if (ret_status == -1) {
            fprintf(stderr,
               "(HyperDoc) Could not find RootPage for top-level window.\n");
            exit(-1);
        }
        /*
         * Tell it how to handle the user defined signals I may get
         */
        bsdSignal(SIGUSR2, sigusr2Handler,RestartSystemCalls);
        bsdSignal(SIGUSR1, SIG_IGN,RestartSystemCalls);
#if defined(BSDplatform) || defined(MACOSXplatform)
        bsdSignal(SIGCHLD, sigcldHandler,RestartSystemCalls);
#else
        bsdSignal(SIGCLD, sigcldHandler,RestartSystemCalls);
#endif
        bsdSignal(SIGINT, SIG_IGN,RestartSystemCalls);
        /*
         * Now go to the main event loop. I will never return, so just end
         * the main routine after that
         */
        /*
         * make an input file if requested
         */
    }
    else {
        /*
         * Try to establish all the socket connections I need. If I am an
         * gIsAxiomServer and the routine fails, it will exit for me
         */
/*        fprintf(stderr,"hyper:main:in else case\n");*/
/*        fprintf(stderr,"hyper:main:calling  makeServerConnections\n");*/
        makeServerConnections();
/*        fprintf(stderr,"hyper:main:returned makeServerConnections\n");*/
        if (make_input_file) ht2Input();
        if (gmakeRecord_file) makeRecord();
        if (gverifyRecord_file) verifyRecord();
        exit(0);
```

```
    }
    /*
     * Try to establish all the socket connections I need. If I am an
     * gIsAxiomServer and the routine fails, it will exit for me
     */
/*    fprintf(stderr,"hyper:main:calling  makeServerConnections\n");*/
    makeServerConnections();
/*    fprintf(stderr,"hyper:main:returned makeServerConnections\n");*/
/*    fprintf(stderr,"hyper:main:calling  mainEventLoop\n");*/
    mainEventLoop();
/*    fprintf(stderr,"hyper:main:returned mainEventLoop\n");*/
    return 0;
}
```

Chapter 10

The htsearch script

Construct a page with a menu of references to the word. The syntax of the command is:

```
htsearch word
```

⟨*htsearch*⟩≡

```
#!/bin/sh

htbindir=$AXIOM/lib
htpagedir=$AXIOM/doc

if test -z "$1"
then
        echo ""|$htbindir/presea case=1 -
else
( cd $htpagedir; $htbindir/hthits "$1" $htpagedir/ht.db | sort -r -n -k 1.22 | $htbindir/pre
fi
```

Chapter 11

The presea script

This is part of 'presea' which is is run on output of 'hthits'. 'hthits' outputs looks like:

```
\newsearchresultentry{1}{Asp24 Example Code}{Asp24ExampleCode}
\newsearchresultentry{1}{Asp27 Example Code}{Asp27ExampleCode}
....
```

after splitting on "{" the first field is `'\newsearchresultentry'` and the second is number of occurences of search term in the page. The test for `'j >= 2'` is just to tolerate garbage. presea is supposed to count the number of matches and put it in the header for search results. The previous version reported no matches in the header. This used to read:

```
        a[n] = $0;
        n=n+1;
        j=split($0,b,"{");
        m=m+substr(b[j],1,length(b[j])-1);
```

⟨*presea*⟩≡

```
#!/bin/awk -f
BEGIN {n=0;m=0
}

{
        a[n] = $0;
        n=n+1;
        j=split($0,b,"{");
        if (j >= 2)
          m=m+substr(b[2],1,length(b[2])-1);
}
```

```
END {
 printf ("\\begin{page}{staticsearchpage}");
 if (case==1)
  printf ("{No matches found}\n")
 else if ( n==0 || m==0 )
   printf ("{No matches found for {\\em %s}}\n",expr)
  else
   printf ("{%d matches found in %d pages for {\\em %s}}\n",m,n,expr);
 printf ("Matches\\tab{8}in Page\n");
 printf "\\beginscroll\n";
 printf "\\beginmenu\n";
 for(i=0;i<n;i++) printf ("%s\n",a[i]);
 printf "\\endmenu\n";
 printf "\\endscroll\n";
 printf "\\end{page}\n";
}
```

11.1 token.h

⟨*token.h*⟩≡

```
/*
 Here are a couple of flags added for whitespace stuff. They tell
      punctuation if there was space in front of it or not
*/

#define FRONTSPACE 0001
#define BACKSPACE  0002



/*
    User tokens. ie, these can be found on a page
*/

#define Word              1
#define Page              2
#define Lispcommandquit   3
#define BoldFace          4
#define Link              5
#define Downlink          6
#define Beginscroll       7
#define Spadcommand       8
#define NoLines           9
#define Env              10
#define Par              11
#define Center           12
#define Begin            13
#define Beginitems       14
#define Item             15
#define Table            16
#define Box              17
#define Tab              18
#define Space            19
#define Indent           20
#define Horizontalline   21
#define Newline          22
#define Enditems         23
#define Returnbutton     24
#define Memolink         25
#define Upbutton         26
#define Endscroll        27
#define Thispage         28
```

```
#define Returnto        29
#define Free            30
#define Bound           31
#define Lisplink        32
#define Unixlink        33
#define Mbox            34
#define Inputstring     35
#define StringValue     36
#define Spadlink        37
#define Inputbitmap     38
#define Inputpixmap     39
#define Unixcommand     40
#define Emphasize       41
#define Lispcommand     42
#define LispMemoLink    43
#define LispDownLink    44
#define Spadcall        45
#define Spadcallquit    46
#define Spaddownlink    47
#define Spadmemolink    48
#define Qspadcall       49
#define Qspadcallquit   50
#define SimpleBox       51
#define Radioboxes      52
#define BoxValue        53
#define VSpace          54
#define HSpace          55
#define NewCommand      56
#define WindowId        57
#define Beep            58
#define Quitbutton      59
#define Begintitems     60
#define Titem           61
#define End             62
#define It              63
#define Sl              64
#define Tt              65
#define Rm              66
#define Ifcond          67
#define Else            68
#define Fi              69
#define Newcond         70
#define Setcond         71
#define Button          72
#define Windowlink      73
#define Haslisp         74
```

```
#define Hasup            75
#define Hasreturn        76
#define Hasreturnto      77
#define Lastwindow       78
#define Endtitems        79
#define Lispwindowlink   80
#define Beginpile        81
#define Endpile          82
#define Nextline         83
#define Pastebutton      84
#define Color            85
#define Helppage         86
#define Patch            87
#define Radiobox         88
#define ifrecond         89
#define Math             90
#define Mitem            91
#define Pagename         92
#define Examplenumber    93
#define Replacepage      94
#define Inputimage       95
#define Spadgraph        96
#define Indentrel        97
#define Controlbitmap    98

#define NumberUserTokens 98

/* places from which input may be read */
#define FromFile       1
#define FromString     2
#define FromSpadSocket 3
#define FromUnixFD     4

extern FILE *unixfd;

/*
 * Here are the system tokens. These are used internally to help
 * with parsing and displaying of text
 */

#define SystemTokens   1001
#define Lbrace         1001
#define Rbrace         1002
#define Macro          1003
#define Group          1004
#define Scrollbar      1005
```

```
#define Pound          1006
#define Lsquarebrace   1007
#define Rsquarebrace   1008
#define Punctuation    1009
#define Dash           1010
#define Tableitem      1011
#define Scrollingnode  1012
#define Headernode     1013
#define Footernode     1014
#define Verbatim       1015
#define Scroll         1016
#define Dollar         1017
#define Percent        1018
#define Carrot         1019
#define Underscore     1020
#define Tilde          1021
#define Cond           1022
#define Noop           1023
#define Description    1024
#define Icorrection    1025
#define Boxcond        1026
#define Unkeyword      1027
#define Titlenode      1028
#define Paste          1029
#define Spadsrc        1030
#define Helpbutton     1031
#define Spadsrctxt     1032


/*
 * Here are the tokens used to mark the end to some sort of group of
 * tokens. ie, the tokens found in a centerline command
 */

#define Endtokens      2000
#define End1           2001
#define End2           2002
#define Endbutton      2003
#define Endlink        2004
#define Endheader      2005
#define Endfooter      2006
#define Endscrolling   2007
#define Endgroup       2008
#define Endarg         2009
#define Endbox         2010
#define Endmbox        2011
```

```
#define Endspadcommand 2012
#define Endpix         2013
#define Endmacro       2014
#define Endparameter   2015
#define Endtable       2016
#define Endtableitem   2017
#define End3           2018
#define Endif          2019
#define Enddescription 2020
#define Endinputbox    2021
#define Endtitle       2022
#define Endpastebutton 2023

#define Endtypes       3000
#define Endpage        3002
#define EndScroll      3007/* use S because Endscroll is already a keyword */

#define Endcenter      3012
#define EndItems       3014 /* use I because Enditems is already a keyword */
#define EndTitems      3060 /* Ibid for the T */
#define Endpatch       3087
#define Endverbatim    4015
#define Endmath        4016
#define Endpaste       4029
#define Endspadsrc     4030
```

Chapter 12

The Bitmaps

12.1 ht_icon

⟨*hticon*⟩≡

```
#define ht_icon_width 40
#define ht_icon_height 40
#define ht_icon_x_hot -1
#define ht_icon_y_hot -1
static char ht_icon_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0xf7, 0x00, 0x00, 0x00, 0x00, 0xe7, 0x00, 0x00, 0x00,
   0x00, 0xe7, 0x00, 0x00, 0x00, 0x00, 0xe7, 0xef, 0x7b, 0x3c, 0xe7, 0xff,
   0xef, 0x7f, 0x7e, 0xff, 0xff, 0xe7, 0xef, 0xe7, 0xfe, 0xe7, 0x6e, 0xe7,
   0xe7, 0xde, 0xe7, 0x7e, 0xe7, 0xff, 0x0e, 0xe7, 0x3c, 0xe7, 0x07, 0x0e,
   0xe7, 0x3c, 0xf7, 0xcf, 0x0e, 0xf7, 0x18, 0x7f, 0xfe, 0x1f, 0x00, 0x1c,
   0x3f, 0x7c, 0x1f, 0x00, 0x0e, 0x07, 0x00, 0x00, 0x00, 0x0f, 0x07, 0x00,
   0x00, 0x00, 0x87, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x80, 0x3f, 0x00, 0x00, 0x00, 0x80, 0x7f, 0x00, 0x00, 0x00,
   0x00, 0x77, 0x00, 0x00, 0x00, 0x00, 0x77, 0x00, 0x00, 0x00, 0x00, 0x77,
   0x00, 0x00, 0x00, 0x00, 0x77, 0x3e, 0xdc, 0x00, 0x00, 0x77, 0x7f, 0xfe,
   0x00, 0x00, 0xf7, 0xe3, 0xef, 0x00, 0x00, 0xf7, 0xe3, 0xc7, 0x00, 0x00,
   0xf7, 0xe3, 0x07, 0x00, 0x00, 0xf7, 0xe3, 0x07, 0x00, 0x00, 0xf7, 0xe3,
   0xcf, 0x00, 0x80, 0x7f, 0x7f, 0xfe, 0x00, 0x80, 0x3f, 0x3e, 0x7c, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

12.2 exit.bitmap

⟨*exit.bitmap*⟩≡

```
#define exit_width 60
#define exit_height 30
#define exit_x_hot -1
#define exit_y_hot -1
static char exit_bits[] = {
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x3f, 0x00, 0xcf, 0x3f,
   0xcf, 0x03, 0xc0, 0xff, 0x3f, 0x00, 0x8e, 0x3f, 0x8e, 0x03, 0x80, 0xff,
   0x3f, 0x00, 0x1e, 0x1f, 0x8f, 0x07, 0x80, 0xff, 0x3f, 0xfe, 0x1f, 0x1f,
   0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0xfe, 0x3f, 0x8e, 0x8f, 0x7f, 0xfc, 0xff,
   0x3f, 0xfe, 0x3f, 0x8e, 0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0xfe, 0x7f, 0xc4,
   0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0xfe, 0x7f, 0xc4, 0x8f, 0x7f, 0xfc, 0xff,
   0x3f, 0xfe, 0xff, 0xe0, 0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0x80, 0xff, 0xe0,
   0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0x00, 0xff, 0xf1, 0x8f, 0x7f, 0xfc, 0xff,
   0x3f, 0x00, 0xff, 0xf1, 0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0xfe, 0xff, 0xe0,
   0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0xfe, 0xff, 0xe0, 0x8f, 0x7f, 0xfc, 0xff,
   0x3f, 0xfe, 0x7f, 0xc4, 0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0xfe, 0x7f, 0xc4,
   0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0xfe, 0x3f, 0x8e, 0x8f, 0x7f, 0xfc, 0xff,
   0x3f, 0xfe, 0x3f, 0x8e, 0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0xfe, 0x1f, 0x1f,
   0x8f, 0x7f, 0xfc, 0xff, 0x3f, 0x00, 0x1f, 0x1f, 0x8f, 0x7f, 0xfc, 0xff,
   0x3f, 0x00, 0x8e, 0x3f, 0x8e, 0x7f, 0xfc, 0xff, 0x7f, 0x00, 0x9e, 0x7f,
   0x9e, 0xff, 0xfc, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```

12.3 help2.bitmap

⟨*help2.bitmap*⟩≡

```
#define help2_width 60
#define help2_height 30
#define help2_x_hot -1
#define help2_y_hot -1
static char help2_bits[] = {
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x9f, 0x9f, 0x07, 0xf0,
   0xfc, 0x0f, 0xf0, 0xff, 0x1f, 0x1f, 0x07, 0xe0, 0xf8, 0x0f, 0xe0, 0xff,
   0x1f, 0x1f, 0x07, 0xe0, 0xf8, 0x0f, 0xc0, 0xff, 0x1f, 0x1f, 0xc7, 0xff,
   0xf8, 0x8f, 0x87, 0xff, 0x1f, 0x1f, 0xc7, 0xff, 0xf8, 0x8f, 0x8f, 0xff,
   0x1f, 0x1f, 0xc7, 0xff, 0xf8, 0x8f, 0x8f, 0xff, 0x1f, 0x1f, 0xc7, 0xff,
   0xf8, 0x8f, 0x8f, 0xff, 0x1f, 0x1f, 0xc7, 0xff, 0xf8, 0x8f, 0x8f, 0xff,
   0x1f, 0x1f, 0xc7, 0xff, 0xf8, 0x8f, 0x8f, 0xff, 0x1f, 0x00, 0x07, 0xf8,
   0xf8, 0x8f, 0x87, 0xff, 0x1f, 0x00, 0x07, 0xf0, 0xf8, 0x0f, 0xc0, 0xff,
   0x1f, 0x00, 0x07, 0xf0, 0xf8, 0x0f, 0xe0, 0xff, 0x1f, 0x1f, 0xc7, 0xff,
   0xf8, 0x0f, 0xf0, 0xff, 0x1f, 0x1f, 0xc7, 0xff, 0xf8, 0x8f, 0xff, 0xff,
   0x1f, 0x1f, 0xc7, 0xff, 0xf8, 0x8f, 0xff, 0xff, 0x1f, 0x1f, 0xc7, 0xff,
   0xf8, 0x8f, 0xff, 0xff, 0x1f, 0x1f, 0xc7, 0xff, 0xf8, 0x8f, 0xff, 0xff,
   0x1f, 0x1f, 0xc7, 0xff, 0xf8, 0x8f, 0xff, 0xff, 0x1f, 0x1f, 0xc7, 0xff,
   0xf8, 0x8f, 0xff, 0xff, 0x1f, 0x1f, 0x07, 0xf0, 0x00, 0x8f, 0xff, 0xff,
   0x1f, 0x1f, 0x07, 0xe0, 0x00, 0x8e, 0xff, 0xff, 0x3f, 0x3f, 0x0f, 0xe0,
   0x01, 0x9c, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```

12.4 return3.bitmap

⟨*return3.bitmap*⟩≡

```
#define return3_width 60
#define return3_height 30
#define return3_x_hot -1
#define return3_y_hot -1
static char return3_bits[] = {
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x9f, 0x9f, 0x0f, 0xf8,
   0xfc, 0x79, 0x00, 0xff, 0x1f, 0x1f, 0x07, 0xf0, 0xf8, 0x71, 0x00, 0xfe,
   0x1f, 0x1f, 0x07, 0xe0, 0xf0, 0x70, 0x00, 0xfe, 0x1f, 0x1f, 0xc7, 0xe3,
   0xf0, 0x70, 0xfc, 0xff, 0x1f, 0x1f, 0xc7, 0xe3, 0xf0, 0x70, 0xfc, 0xff,
   0x1f, 0x1f, 0xc7, 0xe3, 0x60, 0x70, 0xfc, 0xff, 0x1f, 0x1f, 0xc7, 0xe3,
   0x60, 0x70, 0xfc, 0xff, 0x1f, 0x1f, 0xc7, 0xe3, 0x00, 0x70, 0xfc, 0xff,
   0x1f, 0x1f, 0xc7, 0xe3, 0x08, 0x71, 0xfc, 0xff, 0x1f, 0x00, 0xc7, 0xe3,
   0x08, 0x71, 0x80, 0xff, 0x1f, 0x00, 0xc7, 0xe3, 0x98, 0x71, 0x00, 0xff,
   0x1f, 0x00, 0xc7, 0xe3, 0x98, 0x71, 0x00, 0xff, 0x1f, 0x1f, 0xc7, 0xe3,
   0xf8, 0x71, 0xfc, 0xff, 0x1f, 0x1f, 0xc7, 0xe3, 0xf8, 0x71, 0xfc, 0xff,
   0x1f, 0x1f, 0xc7, 0xe3, 0xf8, 0x71, 0xfc, 0xff, 0x1f, 0x1f, 0xc7, 0xe3,
   0xf8, 0x71, 0xfc, 0xff, 0x1f, 0x1f, 0xc7, 0xe3, 0xf8, 0x71, 0xfc, 0xff,
   0x1f, 0x1f, 0xc7, 0xe3, 0xf8, 0x71, 0xfc, 0xff, 0x1f, 0x1f, 0xc7, 0xe3,
   0xf8, 0x71, 0xfc, 0xff, 0x1f, 0x1f, 0x07, 0xe0, 0xf8, 0x71, 0x00, 0xff,
   0x1f, 0x1f, 0x0f, 0xe0, 0xf8, 0x71, 0x00, 0xfe, 0x3f, 0x3f, 0x1f, 0xf0,
   0xf9, 0xf3, 0x00, 0xfe, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```

12.5 up3.bitmap

⟨*up3.bitmap*⟩≡

```
#define up3_width 60
#define up3_height 30
static char up3_bits[] = {
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xdf, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x07,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x01, 0xfc, 0xff, 0xff, 0xff,
   0xff, 0xff, 0x7f, 0x00, 0xf0, 0xff, 0xff, 0xff, 0xff, 0xff, 0x1f, 0x00,
   0xc0, 0xff, 0xff, 0xff, 0xff, 0xff, 0x07, 0x00, 0x00, 0xff, 0xff, 0xff,
   0xff, 0xff, 0x01, 0x00, 0x00, 0xfc, 0xff, 0xff, 0xff, 0xff, 0x3f, 0x00,
   0xe0, 0xff, 0xff, 0xff, 0xff, 0xff, 0x3f, 0x00, 0xe0, 0xff, 0xff, 0xff,
   0xff, 0xff, 0x3f, 0x00, 0xe0, 0xff, 0xff, 0xff, 0xff, 0xff, 0x3f, 0x00,
   0xe0, 0xff, 0xff, 0xff, 0xff, 0xff, 0x3f, 0x00, 0xe0, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```

12.6 noop.bitmap

⟨*noop.bitmap*⟩≡

```
#define noop_width 60
#define noop_height 30
#define noop_x_hot -1
#define noop_y_hot -1
static char noop_bits[] = {
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```

12.7 exit3d.bitmap

⟨*exit3d.bitmap*⟩≡

```
#define exit3d.bitmap_width 60
#define exit3d.bitmap_height 30
static char exit3d.bitmap_bits[] = {
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0c,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0xd1, 0xff, 0x55, 0x55,
   0x5d, 0x55, 0x55, 0x07, 0xaa, 0xff, 0xaa, 0xaa, 0xbe, 0xaa, 0xaa, 0x0e,
   0xd1, 0xd7, 0x55, 0x55, 0x5f, 0xd5, 0x55, 0x07, 0xaa, 0xab, 0xaa, 0xaa,
   0xae, 0xea, 0xaa, 0x0e, 0xd1, 0x57, 0x55, 0x55, 0x55, 0xf5, 0x55, 0x07,
   0xaa, 0xab, 0xaa, 0xaa, 0xaa, 0xea, 0xaa, 0x0e, 0xd1, 0x77, 0x7d, 0x5f,
   0x5f, 0xfd, 0x5f, 0x07, 0xaa, 0xbf, 0xbe, 0xae, 0xbe, 0xfa, 0xaf, 0x0e,
   0xd1, 0x7f, 0x7d, 0x57, 0x5d, 0xf5, 0x55, 0x07, 0xaa, 0xab, 0xfa, 0xab,
   0xbe, 0xea, 0xaa, 0x0e, 0xd1, 0x57, 0xf5, 0x55, 0x5d, 0xf5, 0x55, 0x07,
   0xaa, 0xab, 0xea, 0xab, 0xbe, 0xea, 0xaa, 0x0e, 0xd1, 0x57, 0xd5, 0x57,
   0x5d, 0xf5, 0x55, 0x07, 0xaa, 0xab, 0xea, 0xaf, 0xbe, 0xea, 0xaa, 0x0e,
   0xd1, 0xd7, 0x75, 0x5f, 0x5d, 0xf5, 0x55, 0x07, 0xaa, 0xff, 0xba, 0xbe,
   0xbe, 0xea, 0xaf, 0x0e, 0xd1, 0xff, 0x7d, 0x5f, 0x7f, 0xd5, 0x57, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e,
   0xfd, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x07, 0xfe, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0x0f, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x05};
```

12.8 help3d.bitmap

⟨*help3d.bitmap*⟩≡

```
#define help3d.bitmap_width 60
#define help3d.bitmap_height 30
static char help3d.bitmap_bits[] = {
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0c,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0xd1, 0xf7, 0x55, 0x55,
   0x5f, 0x55, 0x55, 0x07, 0xaa, 0xeb, 0xaa, 0xaa, 0xbe, 0xaa, 0xaa, 0x0e,
   0xd1, 0xf7, 0x55, 0x55, 0x5d, 0x55, 0x55, 0x07, 0xaa, 0xeb, 0xaa, 0xaa,
   0xbe, 0xaa, 0xaa, 0x0e, 0xd1, 0xf7, 0x55, 0x55, 0x5d, 0x55, 0x55, 0x07,
   0xaa, 0xeb, 0xaa, 0xaa, 0xbe, 0xaa, 0xaa, 0x0e, 0xd1, 0xf7, 0xf5, 0x57,
   0x5d, 0xdd, 0x57, 0x07, 0xaa, 0xff, 0xfa, 0xaf, 0xbe, 0xfa, 0xaf, 0x0e,
   0xd1, 0xff, 0x7d, 0x5f, 0x5d, 0x7d, 0x5f, 0x07, 0xaa, 0xeb, 0xbe, 0xae,
   0xbe, 0xba, 0xbe, 0x0e, 0xd1, 0xf7, 0xfd, 0x5f, 0x5d, 0x7d, 0x5d, 0x07,
   0xaa, 0xeb, 0xfe, 0xaf, 0xbe, 0xba, 0xbe, 0x0e, 0xd1, 0xf7, 0x5d, 0x55,
   0x5d, 0x7d, 0x5d, 0x07, 0xaa, 0xeb, 0xbe, 0xaa, 0xbe, 0xba, 0xbe, 0x0e,
   0xd1, 0xf7, 0x7d, 0x5d, 0x5d, 0x7d, 0x5f, 0x07, 0xaa, 0xeb, 0xfa, 0xaf,
   0xbe, 0xfa, 0xaf, 0x0e, 0xd1, 0xf7, 0xf5, 0x57, 0x7f, 0xfd, 0x57, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xba, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55,
   0x55, 0x7d, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xba, 0xaa, 0x0e,
   0xf9, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x07, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0x0f, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x05};
```

12.9 home3d.bitmap

⟨*home3d.bitmap*⟩≡

```
#define home3d.bitmap_width 60
#define home3d.bitmap_height 30
static char home3d.bitmap_bits[] = {
   0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x05, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xef, 0xab, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0xd7, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xaa, 0xef, 0xab, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0xd7, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0xaa, 0xef, 0xab, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e,
   0x51, 0xd7, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xef, 0xeb, 0xaf,
   0xbb, 0xeb, 0xaf, 0x0e, 0x51, 0xff, 0xf5, 0xdf, 0xff, 0xf7, 0x5f, 0x07,
   0xaa, 0xff, 0xfb, 0xae, 0xbb, 0xfb, 0xbe, 0x0e, 0x51, 0xd7, 0x7d, 0xdd,
   0xff, 0x7f, 0x5d, 0x07, 0xaa, 0xef, 0xbb, 0xbe, 0xbb, 0xfb, 0xbf, 0x0e,
   0x51, 0xd7, 0x7d, 0xdd, 0xff, 0xff, 0x5f, 0x07, 0xaa, 0xef, 0xbb, 0xbe,
   0xbb, 0xbb, 0xaa, 0x0e, 0x51, 0xd7, 0x7d, 0xdd, 0xff, 0x7f, 0x55, 0x07,
   0xaa, 0xef, 0xfb, 0xae, 0xbb, 0xfb, 0xba, 0x0e, 0x51, 0xd7, 0xf5, 0xdf,
   0xff, 0xf7, 0x5f, 0x07, 0xaa, 0xef, 0xeb, 0xaf, 0xbb, 0xeb, 0xaf, 0x0e,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xfa, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x0f, 0xff, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0a};
```

12.10 up3d.bitmap

⟨*up3d.bitmap*⟩≡

```
#define up3_width 60
#define up3_height 30
static char up3_bits[] = {
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x05, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0c,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xfa,
   0xab, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0xfd, 0x57, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xff, 0xbf, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0xd5, 0xff,
   0x7f, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xfa, 0xff, 0xff, 0xab, 0xaa, 0x0e,
   0x51, 0x55, 0xfd, 0xff, 0xff, 0x57, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xff,
   0xbf, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0xd5, 0xff, 0x7f, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xff, 0xbf, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0xd5, 0xff,
   0x7f, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xff, 0xbf, 0xaa, 0xaa, 0x0e,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e,
   0xf9, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x07, 0xfe, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0x0f, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x05};
```

12.11 noop3d.bitmap

⟨*noop3d.bitmap*⟩≡

```
#define noop_width 60
#define noop_height 30
static char noop_bits[] = {
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0a, 0x55, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x05, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0c,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e,
   0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa,
   0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x07,
   0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e, 0x51, 0x55, 0x55, 0x55,
   0x55, 0x55, 0x55, 0x07, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0x0e,
   0xf9, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x07, 0xfe, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0x0f, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x05};
```

Chapter 13

Makefile

⟨*⟩≡

```
BOOK=${SPD}/books/bookvol7.pamphlet
WORK=${OBJ}/${SYS}/hyper
IN=${SPD}/books

# this is where to put the various commands
OUTBIN= ${MNT}/${SYS}/bin
OUTLIB= ${MNT}/${SYS}/lib

# this is where the include files live
INC=    ${SRC}/include

# this is where we hid the libspad library
LIB= ${OBJ}/${SYS}/lib

HYPER=${MNT}/${SYS}/doc

CFLAGS= ${CCF}
LDFLAGS= -L${LIB} -lspad ${LDF}

LIBS = ${LIB}/sockio-c.o ${LIB}/bsdsignal.o

SPADBUF_LIBS=${LIBS} ${LIB}/wct.o ${LIB}/edin.o ${LIB}/prt.o \
             ${LIB}/cursor.o ${LIB}/fnct-key.o

HYPER_LIBS=${LIBS} ${LIB}/pixmap.o ${LIB}/spadcolors.o ${LIB}/util.o

all: ${OUTLIB}/spadbuf ${OUTLIB}/ex2ht ${OUTBIN}/htadd ${OUTLIB}/hthits \
     ${OUTLIB}/htsearch ${OUTLIB}/presea ${OUTBIN}/hypertex \
     ${HYPER}/axbook ${HYPER}/bigbayou.png ${HYPER}/doctitle.png
```

```
        @cp -pr ${IN}/bitmaps ${HYPER}
        @ echo 0 finished ${BOOK}

${OUTLIB}/spadbuf: ${BOOK}
        @ echo 1 making ${OUTLIB}/spadbuf from ${BOOK}
        @ (cd ${WORK} ; \
           ${TANGLE} -R"spadbuf" ${BOOK} >spadbuf.c ; \
           ${CC} -I${INC} ${CFLAGS} spadbuf.c -o ${OUTLIB}/spadbuf ${LDFLAGS} )

${OUTLIB}/ex2ht: ${BOOK}
        @ echo 2 making ${OUTLIB}/ex2ht from ${BOOK}
        @ (cd ${WORK} ; \
            ${TANGLE} -R"ex2ht" ${BOOK} >ex2ht.c ; \
            ${CC} -I${INC} ${CFLAGS} ex2ht.c -o ${OUTLIB}/ex2ht ${LDFLAGS} )

${OUTBIN}/htadd: ${BOOK}
        @ echo 3 making ${OUTBIN}/htadd from ${BOOK}
        @ (cd ${WORK} ; \
          ${TANGLE} -R"htadd" ${BOOK} >htadd.c ; \
          ${CC} -I${INC} ${CFLAGS} htadd.c -o ${OUTBIN}/htadd ${LDFLAGS} )

${OUTLIB}/hthits: ${BOOK}
        @ echo 4 making ${OUTBIN}/hthits from ${BOOK}
        @ (cd ${WORK} ; \
          ${TANGLE} -R"hthits" ${BOOK} >hthits.c ; \
          ${CC} -I${INC} ${CFLAGS} hthits.c -o ${OUTLIB}/hthits ${LDFLAGS} )

${OUTLIB}/htsearch: ${BOOK}
        @echo 5 making ${OUTLIB}/htsearch from ${BOOK}
        @${TANGLE} -R"htsearch" ${BOOK} >${OUTLIB}/htsearch
        @chmod a+x ${OUTLIB}/htsearch

${OUTLIB}/presea: ${BOOK}
        @echo 6 making ${OUTLIB}/presea from ${BOOK}
        @${TANGLE} -R"presea" ${BOOK} >${OUTLIB}/presea
        @chmod a+x ${OUTLIB}/presea

${OUTBIN}/hypertex: ${BOOK}
        @ echo 7 ${OUTBIN}/hypertex from ${BOOK}
        @ (cd ${WORK} ; \
          ${TANGLE} -R"hypertex" ${BOOK} >hypertex.c ; \
          ${CC} -I${INC} ${CFLAGS} hypertex.c -o ${OUTBIN}/hypertex \
                  ${LDFLAGS} -lXpm -lX11 -lm -L${LIB} )

${HYPER}/bigbayou.png: ${IN}/ps/bigbayou.png
        @ echo 8 making ${HYPER}/bigbayou.png from ${IN}/bigbayou.png
```

```
        @ cp ${IN}/ps/bigbayou.png ${HYPER}/bigbayou.png

${HYPER}/doctitle.png: ${IN}/ps/doctitle.png
        @ echo 9 making ${HYPER}/doctitle.png from ${IN}/doctitle.png
        @ cp ${IN}/ps/doctitle.png ${HYPER}/doctitle.png

${HYPER}/axbook: ${IN}/axbook.tgz
        @ echo 10 making ${HYPER}/axbook/xhtml from ${IN}/axbook.tgz
        @( cd ${HYPER} ; tar -zxf ${IN}/axbook.tgz )
```

Bibliography

[1] Jenks, R.J. and Sutor, R.S. "Axiom – The Scientific Computation System" Springer-Verlag New York (1992) ISBN 0-387-97855-0

[2] Knuth, Donald E., "Literate Programming" Center for the Study of Language and Information ISBN 0-937073-81-4 Stanford CA (1992)

[3] Daly, Timothy, "The Axiom Wiki Website"
http://axiom.axiom-developer.org

[4] Watt, Stephen, "Aldor",
http://www.aldor.org

[5] Lamport, Leslie, "Latex – A Document Preparation System", Addison-Wesley, New York ISBN 0-201-52983-1

[6] Ramsey, Norman "Noweb – A Simple, Extensible Tool for Literate Programming"
http://www.eecs.harvard.edu/ ̃nr/noweb

[7] Daly, Timothy, "The Axiom Literate Documentation"
http://axiom.axiom-developer.org/axiom-website/documentation.html

Index

acceptMenuConnection hypertex, 499
acceptMenuServerConnection hypertex, 500
addBoxToRbList hypertex, 425
addBufferToSym hypertex, 179
addDependencies hypertex, 410
addFile ex2ht, 110
addfile htadd, 119
addNewPages htadd, 123
allocButtonList hypertex, 377
allocCondnode hypertex, 368
allocHdWindow hypertex, 361
allocIfnode hypertex, 367
allocInputbox hypertex, 377
allocInputline hypertex, 375
allocNode hypertex, 363
allocPage hypertex, 369
allocPasteNode hypertex, 376
allocPatchstore hypertex, 376
allocRbs hypertex, 377
allocString ex2ht, 104
allocString shared, 43
alreadyThere hypertex, 428

backOverChar hypertex, 195
backOverEoln hypertex, 191
badDB hthits, 138
beginType shared, 64
BeStruct struct, 31
beType shared, 62
bfTopGroup hypertex, 299
buildDBFilename htadd, 117
buildHtFilename shared, 33

calculateScrollBarMeasures hypertex, 472
centerNodes hypertex, 273
centerTopGroup hypertex, 300
ch hypertex, 478
changeCond hypertex, 168
changeCursor hypertex, 236
changeInputFocus hypertex, 340
changeText hypertex, 334
changeWindowBackgroundPixmap hypertex, 478
checkAndPopBeStack shared, 61
checkArguments hypertex, 164
checkCondition hypertex, 170
checkMemostack hypertex, 169
checkOthers hypertex, 425
cleanSocket hypertex, 162
clearBeStack shared, 61
clearCursor hypertex, 186
clearCursorline hypertex, 176
clearExecutionMarks hypertex, 498
clearExposures hypertex, 232
clearItemStack hypertex, 344
clearRbs hypertex, 340
closeClient hypertex, 513
closeCoverFile ex2ht, 109
closeCoverPage ex2ht, 109
cmdline hthits, 131
computeBeginItemsExtent hypertex, 252
computeBfExtent hypertex, 256
computeBoxExtent hypertex, 265
computeButtonExtent hypertex, 257
computeCenterExtent hypertex, 255
computeDashExtent hypertex, 243
computeEmExtent hypertex, 256
computeFooterExtent hypertex, 271
computeFormPage hypertex, 294
computeHeaderExtent hypertex, 270
computeIfcondExtent hypertex, 254

computeImageExtent hypertex, 267
computeInputExtent hypertex, 238
computeIrExtent hypertex, 266
computeItemExtent hypertex, 253
computeItExtent hypertex, 256
computeMboxExtent hypertex, 264
computeMitemExtent hypertex, 253
computePastebuttonExtent hypertex, 259
computePasteExtent hypertex, 260
computePunctuationExtent hypertex, 239
computeRmExtent hypertex, 257
computeScrollingExtent hypertex, 272
computeSpadcommandExtent hypertex, 261
computeSpadsrcExtent hypertex, 262
computeSpadsrctxtExtent hypertex, 242
computeTableExtent hypertex, 268
computeTextExtent hypertex, 245
computeTitleExtent hypertex, 269
computeVerbatimExtent hypertex, 242
computeWordExtent hypertex, 241
connectSpad shared, 69
copyFile htadd, 124
copyGroupStack hypertex, 301
copyItemStack hypertex, 345
createWindow hypertex, 217
currentItem hypertex, 427

dbFileOpen shared, 36
decLineNumbers hypertex, 172
decreaseLineNumbers hypertex, 172
deleteChar hypertex, 199
deleteDB htadd, 127
deleteEoln hypertex, 196
deleteFile htadd, 126
deleteItem hypertex, 343
deleteOneChar hypertex, 198
deleteRestOfLine hypertex, 189
dialog hypertex, 204
displayPage hypertex, 381
dontFree hypertex, 373
downlink hypertex, 219
drawCursor hypertex, 181
drawInputsymbol hypertex, 179
drawScroller3DEffects hypertex, 468
drawScrollLines hypertex, 471
dumpToken shared, 44

emitCoverLink ex2ht, 109
emitFooter ex2ht, 107
emitHeader ex2ht, 107
emitMenuEntry ex2ht, 108
emitSpadCommand ex2ht, 108
emTopGroup hypertex, 298
endAPage hypertex, 399
endbuttonExtent hypertex, 258
endifExtent hypertex, 253
endpastebuttonExtent hypertex, 260
endSpadcommandExtent hypertex, 263
endSpadsrcExtent hypertex, 264
endType shared, 65
enterNewLine hypertex, 202
escapeString hypertex, 512
ex2ht
    addFile, 110
    allocString, 104
    closeCoverFile, 109
    closeCoverPage, 109
    emitCoverLink, 109
    emitFooter, 107
    emitHeader, 107
    emitMenuEntry, 108
    emitSpadCommand, 108
    exToHt, 106
    getExTitle, 105
    main, 110
    openCoverPage, 109
    strPrefix, 104
exitHyperDoc hypertex, 230
exposePage hypertex, 209
extendHT shared, 32
exToHt ex2ht, 106

fillBox hypertex, 338
findButtonInList hypertex, 224
findFp hypertex, 415
findPage hypertex, 219
formatPage hypertex, 382

formFooterExtent hypertex, 296
formHeaderExtent hypertex, 295
formScrollingExtent hypertex, 296
freeButtonList hypertex, 378
freeCond hypertex, 368
freeDepend hypertex, 373
freeGroupStack hypertex, 301
freeHash shared, 39
freeHdWindow hypertex, 362
freeIfNonNULL hypertex, 360
freeInputBox hypertex, 375
freeInputItem hypertex, 374
freeInputList hypertex, 374
freeItemStack hypertex, 346
freeLines hypertex, 374
freeNode hypertex, 364
freePage hypertex, 370
freePaste hypertex, 371
freePastearea hypertex, 373
freePastebutton hypertex, 372
freePatch hypertex, 376
freeRadioBoxes hypertex, 375
freeString hypertex, 373

getBorderProperties hypertex, 324
getChar shared, 53
getChar1 shared, 55
getColor hypertex, 335
getExpectedToken shared, 67
getExTitle ex2ht, 105
getFilename htadd, 125
getFilename hypertex, 412
getGCs hypertex, 328
getGraphOutput hypertex, 312
getHyperLink hypertex, 225
getInputString hypertex, 413
getModifierMask hypertex, 350
getNewWindow hypertex, 233
getParameterStrings hypertex, 358
getScrollBarMinimumSize hypertex, 478
getSpadOutput hypertex, 312
getTitleBarMinimumSize hypertex, 529
getToken shared, 57
getWhere hypertex, 414

halloc shared, 38
handleButton hypertex, 226
handleEvent hypertex, 215
handleFile hthits, 132
handleFilePages hthits, 134
handleHtdb hthits, 131
handleKey hypertex, 347
handleMotionEvent hypertex, 236
handlePage hthits, 135
hashCopyEntry shared, 42
hashCopyTable shared, 42
hashDelete shared, 41
hashFind shared, 40
hashInit shared, 39
hashInsert shared, 40
hashMap shared, 41
hashReplace shared, 41
helpForHyperDoc hypertex, 224
hideScrollBars hypertex, 478
ht2Input hypertex, 303
htadd
    addfile, 119
    addNewPages, 123
    buildDBFilename, 117
    copyFile, 124
    deleteDB, 127
    deleteFile, 126
    getFilename, 125
    main, 128
    parseArgs, 115
    updateDB, 121
    writable, 116
htadd shared code, 70
htFileOpen
    main, 35
hthits
    badDB, 138
    cmdline, 131
    handleFile, 132
    handleFilePages, 134
    handleHtdb, 131
    handlePage, 135
    main, 139
    regerr, 139
    searchPage, 136
    splitpage, 137

squirt, 136
untexbuf, 138
htperror shared, 49
HTReadBitmapFile hypertex, 460
HyperDocErrorHandler hypertex, 237
hypertex, 141
acceptMenuConnection, 499
acceptMenuServerConnection, 500
addBoxToRbList, 425
addBufferToSym, 179
addDependencies, 410
allocButtonList, 377
allocCondnode, 368
allocHdWindow, 361
allocIfnode, 367
allocInputbox, 377
allocInputline, 375
allocNode, 363
allocPage, 369
allocPasteNode, 376
allocPatchstore, 376
allocRbs, 377
alreadyThere, 428
backOverChar, 195
backOverEoln, 191
bfTopGroup, 299
calculateScrollBarMeasures, 472
centerNodes, 273
centerTopGroup, 300
ch, 478
changeCond, 168
changeCursor, 236
changeInputFocus, 340
changeText, 334
changeWindowBackgroundPixmap, 478
checkArguments, 164
checkCondition, 170
checkMemostack, 169
checkOthers, 425
cleanSocket, 162
clearCursor, 186
clearCursorline, 176
clearExecutionMarks, 498
clearExposures, 232
clearItemStack, 344
clearRbs, 340
closeClient, 513
computeBeginItemsExtent, 252
computeBfExtent, 256
computeBoxExtent, 265
computeButtonExtent, 257
computeCenterExtent, 255
computeDashExtent, 243
computeEmExtent, 256
computeFooterExtent, 271
computeFormPage, 294
computeHeaderExtent, 270
computeIfcondExtent, 254
computeImageExtent, 267
computeInputExtent, 238
computeIrExtent, 266
computeItemExtent, 253
computeItExtent, 256
computeMboxExtent, 264
computeMitemExtent, 253
computePastebuttonExtent, 259
computePasteExtent, 260
computePunctuationExtent, 239
computeRmExtent, 257
computeScrollingExtent, 272
computeSpadcommandExtent, 261
computeSpadsrcExtent, 262
computeSpadsrctxtExtent, 242
computeTableExtent, 268
computeTextExtent, 245
computeTitleExtent, 269
computeVerbatimExtent, 242
computeWordExtent, 241
copyGroupStack, 301
copyItemStack, 345
createWindow, 217
currentItem, 427
decLineNumbers, 172
decreaseLineNumbers, 172
deleteChar, 199
deleteEoln, 196
deleteItem, 343
deleteOneChar, 198
deleteRestOfLine, 189
dialog, 204
displayPage, 381

dontFree, 373
downlink, 219
drawCursor, 181
drawInputsymbol, 179
drawScroller3DEffects, 468
drawScrollLines, 471
emTopGroup, 298
endAPage, 399
endbuttonExtent, 258
endifExtent, 253
endpastebuttonExtent, 260
endSpadcommandExtent, 263
endSpadsrcExtent, 264
enterNewLine, 202
escapeString, 512
exitHyperDoc, 230
exposePage, 209
fillBox, 338
findButtonInList, 224
findFp, 415
findPage, 219
formatPage, 382
formFooterExtent, 296
formHeaderExtent, 295
formScrollingExtent, 296
freeButtonList, 378
freeCond, 368
freeDepend, 373
freeGroupStack, 301
freeHdWindow, 362
freeIfNonNULL, 360
freeInputBox, 375
freeInputItem, 374
freeInputList, 374
freeItemStack, 346
freeLines, 374
freeNode, 364
freePage, 370
freePaste, 371
freePastearea, 373
freePastebutton, 372
freePatch, 376
freeRadioBoxes, 375
freeString, 373
getBorderProperties, 324
getColor, 335
getFilename, 412
getGCs, 328
getGraphOutput, 312
getHyperLink, 225
getInputString, 413
getModifierMask, 350
getNewWindow, 233
getParameterStrings, 358
getScrollBarMinimumSize, 478
getSpadOutput, 312
getTitleBarMinimumSize, 529
getWhere, 414
handleButton, 226
handleEvent, 215
handleKey, 347
handleMotionEvent, 236
helpForHyperDoc, 224
hideScrollBars, 478
ht2Input, 303
HTReadBitmapFile, 460
HyperDocErrorHandler, 237
incLineNumbers, 171
ingItColorsAndFonts, 330
initCursorState, 236
initCursorStates, 237
initExtents, 283
initFormWindow, 322
initGroupStack, 298
initHash, 163
initializeDefault, 418
initializeWindowSystem, 316
initKeyin, 351
initPageStructs, 163
initParameterElem, 355
initParsePage, 385
initParsePatch, 385
initText, 284
initTitleExtents, 283
initTopGroup, 300
initTopWindow, 318
inListAndNewer, 309
inputStringWidth, 274
insertBitmapFile, 292
insertBuffer, 177
insertCond, 168
insertImageStruct, 464

insertItem, 426
insertPixmapFile, 293
isIt850, 338
isNumber, 411
issueDependentCommands, 495
issueServerCommand, 508
issueServerpaste, 509
issueSpadcommand, 493
issueUnixcommand, 509
issueUnixlink, 510
issueUnixpaste, 510
killAxiomPage, 220
killPage, 220
lineTopGroup, 299
linkScrollBars, 473
linkTitleBarWindows, 527
lispwindowlinkHandler, 222
loadFont, 329
loadMacro, 354
loadPage, 380
loadPatch, 438
main, 409, 530
mainEventLoop, 213
makeBoxWindow, 417
makeBusyCursor, 237
makeBusyCursors, 237
makeInputFileFromPage, 306
makeInputFileList, 310
makeInputFileName, 304
makeInputWindow, 416
makeLinkWindow, 407
makePasteFileName, 304
makePasteWindow, 408
makeRecord, 302
makeScrollBarWindows, 466
makeServerConnections, 166
makeSpecialPage, 409
makeTheInputFile, 305
makeTitleBarWindows, 524
makeWindowLink, 222
markAsExecuted, 496
maxX, 288
memolink, 220
mergeDatabases, 337
moveBackOneChar, 193
moveCursorBackward, 187
moveCursorDown, 184
moveCursorEnd, 182
moveCursorHome, 182
moveCursorUp, 185
moveRestBack, 188
moveScroller, 470
moveSymForward, 175
mystrncpy, 171
nextInputFocus, 341
number, 353
openFormWindow, 320
openWindow, 325
overwriteBuffer, 173
parseBeginItems, 444
parseBox, 458
parseButton, 451
parseCenterline, 449
parseCommand, 450
parseCondnode, 441
parseEnv, 454
parseFree, 459
parseFromString, 382
parseHasreturnto, 442
parseHeader, 384
parseHelp, 459
parseHyperDoc, 387
parseIfcond, 439
parseInputPix, 448
parseInputstring, 419
parseItem, 445
parseMacro, 357
parseMbox, 458
parseMitem, 446
parseNewcond, 442
parsePage, 386
parsePageFromSocket, 395
parsePageFromUnixfd, 396
parseParameters, 360
parsePaste, 431
parsePastebutton, 433
parsePatch, 435
parseRadiobox, 423
parseRadioboxes, 429
parseReplacepage, 399
parserError, 411
parseSetcond, 443

parseSimplebox, 421
parseSpadcommand, 452
parseSpadsrc, 453
parseTable, 457
parseTitle, 383
parseValue1, 455
parseValue2, 456
parseVerbatim, 447
pasteButton, 223
pastePage, 211
plh, 294
popGroupStack, 297
popItemStack, 345
PopMR, 379
popParameters, 356
prevInputFocus, 342
printGraphPaste, 315
printPaste, 314
printPasteLine, 311
printSourceToString, 514
printSourceToString1, 515
printToString, 501
printToString1, 502
punctuationWidth, 274
pushActiveGroup, 300
pushGroupStack, 297
pushItemStack, 344
PushMR, 379
pushParameters, 356
pushSpadGroup, 300
quitHyperDoc, 218
readHot, 462
readHtDb, 401
readHtFile, 403
readTitleBarImages, 528
readWandH, 463
redrawWin, 171
repasteItem, 427
resizeBuffer, 378
returnItem, 343
returnlink, 221
rmTopGroup, 298
scanHyperDoc, 352
scrollDown, 475
scrollDownPage, 476
scrollPage, 210
scrollScroller, 477
scrollToFirstPage, 475
scrollUp, 474
scrollUpPage, 474
sendCommand, 313
sendLispCommand, 512
sendPile, 494
serviceSessionSocket, 511
setCursor, 235
setErrorHandlers, 238
setNameAndIcon, 323
setSizeHints, 326
setWindow, 231
showImage, 491
showInput, 488
showLink, 486
showPage, 207
showPaste, 487
showPastebutton, 488
showScrollBars, 469
showSimpleBox, 489
showSpadcommand, 490
showText, 480
showTitleBar, 525
sigcldHandler, 162
sigusr2Handler, 162
startFooter, 398
startNewline, 273
startScrolling, 397
startUserBuffer, 497
strCopy, 308
switchFrames, 511
textHeight, 284
textHeight1, 285
textWidth, 276
toggleInputBox, 339
toggleRadioBox, 339
totalWidth, 281
toughEnter, 200
trailingSpace, 291
ttTopGroup, 299
unescapeString, 512
updateInputsymbol, 180
uplink, 221
verbatimWidth, 275
verifyRecord, 303

void moveCursorForward, 183
widthOfDash, 275
windowCode, 400
windowEqual, 400
windowHeight, 295
windowId, 400
windowlinkHandler, 222
windowWidth, 295
wordWidth, 274
Xvalue, 290
hypertex shared code, 75

incLineNumbers hypertex, 171
ingItColorsAndFonts hypertex, 330
initCursorState hypertex, 236
initCursorStates hypertex, 237
initExtents hypertex, 283
initFormWindow hypertex, 322
initGroupStack hypertex, 298
initHash hypertex, 163
initializeDefault hypertex, 418
initializeWindowSystem hypertex, 316
initKeyin hypertex, 351
initPageStructs hypertex, 163
initParameterElem hypertex, 355
initParent spadbuf, 100
initParsePage hypertex, 385
initParsePatch hypertex, 385
initPasteItem hypertex, 426
initScanner shared, 51
initText hypertex, 284
initTitleExtents hypertex, 283
initTopGroup hypertex, 300
initTopWindow hypertex, 318
inListAndNewer hypertex, 309
inputStringWidth hypertex, 274
insertBitmapFile hypertex, 292
insertBuffer hypertex, 177
insertCond hypertex, 168
insertImageStruct hypertex, 464
insertItem hypertex, 426
insertPixmapFile hypertex, 293
interpIO spadbuf, 99
isIt850 hypertex, 338
isNumber hypertex, 411
issueDependentCommands hypertex, 495
issueServerCommand hypertex, 508
issueServerpaste hypertex, 509
issueSpadcommand hypertex, 493
issueUnixcommand hypertex, 509
issueUnixlink hypertex, 510
issueUnixpaste hypertex, 510

jump shared, 44

keywordType shared, 66
killAxiomPage hypertex, 220
killPage hypertex, 220

lineTopGroup hypertex, 299
linkScrollBars hypertex, 473
linkTitleBarWindows hypertex, 527
lispwindowlinkHandler hypertex, 222
loadFont hypertex, 329
loadMacro hypertex, 354
loadPage hypertex, 380
loadPatch hypertex, 438

main ex2ht, 110
main htadd, 128
main htFileOpen, 35
main hthits, 139
main hypertex, 409, 530
main spadbuf, 101
mainEventLoop hypertex, 213
makeBoxWindow hypertex, 417
makeBusyCursor hypertex, 237
makeBusyCursors hypertex, 237
makeInputFileFromPage hypertex, 306
makeInputFileList hypertex, 310
makeInputFileName hypertex, 304
makeInputWindow hypertex, 416
makeLinkWindow hypertex, 407
makePasteFileName hypertex, 304
makePasteWindow hypertex, 408
makeRecord hypertex, 302
makeScrollBarWindows hypertex, 466
makeServerConnections hypertex, 166
makeSpecialPage hypertex, 409
makeTheInputFile hypertex, 305

makeTitleBarWindows hypertex, 524
makeWindowLink hypertex, 222
markAsExecuted hypertex, 496
maxX hypertex, 288
memolink hypertex, 220
mergeDatabases hypertex, 337
moveBackOneChar hypertex, 193
moveCursorBackward hypertex, 187
moveCursorDown hypertex, 184
moveCursorEnd hypertex, 182
moveCursorHome hypertex, 182
moveCursorUp hypertex, 185
moveRestBack hypertex, 188
moveScroller hypertex, 470
moveSymForward hypertex, 175
mystrncpy hypertex, 171

nextInputFocus hypertex, 341
number hypertex, 353

openCoverPage ex2ht, 109
openFormWindow hypertex, 320
openWindow hypertex, 325
overwriteBuffer hypertex, 173

parseArgs htadd, 115
parseBeginItems hypertex, 444
parseBox hypertex, 458
parseButton hypertex, 451
parseCenterline hypertex, 449
parseCommand hypertex, 450
parseCondnode hypertex, 441
parseEnv hypertex, 454
parseFree hypertex, 459
parseFromString hypertex, 382
parseHasreturnto hypertex, 442
parseHeader hypertex, 384
parseHelp hypertex, 459
parseHyperDoc hypertex, 387
parseIfcond hypertex, 439
parseInputPix hypertex, 448
parseInputstring hypertex, 419
parseItem hypertex, 445
parseMacro hypertex, 357
parseMbox hypertex, 458
parseMitem hypertex, 446
parseNewcond hypertex, 442
parsePage hypertex, 386
parsePageFromSocket hypertex, 395
parsePageFromUnixfd hypertex, 396
parseParameters hypertex, 360
parsePaste hypertex, 431
parsePastebutton hypertex, 433
parsePatch hypertex, 435
parseRadiobox hypertex, 423
parseRadioboxes hypertex, 429
parseReplacepage hypertex, 399
parserError hypertex, 411
parserInit shared, 50
parseSetcond hypertex, 443
parseSimplebox hypertex, 421
parseSpadcommand hypertex, 452
parseSpadsrc hypertex, 453
parseTable hypertex, 457
parseTitle hypertex, 383
parseValue1 hypertex, 455
parseValue2 hypertex, 456
parseVerbatim hypertex, 447
pasteButton hypertex, 223
pastePage hypertex, 211
pathname shared, 35
PgInfo struct, 130
plh hypertex, 294
popGroupStack hypertex, 297
popItemStack hypertex, 345
PopMR hypertex, 379
popParameters hypertex, 356
prevInputFocus hypertex, 342
printGraphPaste hypertex, 315
printNextTenTokens shared, 46
printPageAndFilename shared, 45
printPaste hypertex, 314
printPasteLine hypertex, 311
printSourceToString hypertex, 514
printSourceToString1 hypertex, 515
printToken shared, 46
printToString hypertex, 501
printToString1 hypertex, 502
punctuationWidth hypertex, 274
pushActiveGroup hypertex, 300
pushBeStack shared, 60
pushGroupStack hypertex, 297

pushItemStack hypertex, 344
PushMR hypertex, 379
pushParameters hypertex, 356
pushSpadGroup hypertex, 300

quitHyperDoc hypertex, 218

readHot hypertex, 462
readHtDb hypertex, 401
readHtFile hypertex, 403
readTitleBarImages hypertex, 528
readWandH hypertex, 463
redrawWin hypertex, 171
regerr hthits, 139
repasteItem hypertex, 427
resetConnection shared, 68
resizeBuffer hypertex, 378
restoreScannerState shared, 52
returnItem hypertex, 343
returnlink hypertex, 221
rmTopGroup hypertex, 298

saveScannerState shared, 51
scanHyperDoc hypertex, 352
scrollDown hypertex, 475
scrollDownPage hypertex, 476
scrollPage hypertex, 210
scrollScroller hypertex, 477
scrollToFirstPage hypertex, 475
scrollUp hypertex, 474
scrollUpPage hypertex, 474
searchPage hthits, 136
sendCommand hypertex, 313
sendLispCommand hypertex, 512
sendPile hypertex, 494
serviceSessionSocket hypertex, 511
setCursor hypertex, 235
setErrorHandlers hypertex, 238
setNameAndIcon hypertex, 323
setSizeHints hypertex, 326
setWindow hypertex, 231
shared
    allocString, 43
    beginType, 64
    beType, 62
    buildHtFilename, 33
    checkAndPopBeStack, 61
    clearBeStack, 61
    connectSpad, 69
    dbFileOpen, 36
    dumpToken, 44
    endType, 65
    extendHT, 32
    freeHash, 39
    getChar, 53
    getChar1, 55
    getExpectedToken, 67
    getToken, 57
    halloc, 38
    hashCopyEntry, 42
    hashCopyTable, 42
    hashDelete, 41
    hashFind, 40
    hashInit, 39
    hashInsert, 40
    hashMap, 41
    hashReplace, 41
    htperror, 49
    initScanner, 51
    jump, 44
    keywordType, 66
    parserInit, 50
    pathname, 35
    printNextTenTokens, 46
    printPageAndFilename, 45
    printToken, 46
    pushBeStack, 60
    resetConnection, 68
    restoreScannerState, 52
    saveScannerState, 51
    spadBusy, 68
    spadErrorHandler, 67
    stringEqual, 43
    stringHash, 43
    strpostfix, 31
    tempFileOpen, 37
    tokenName, 47
    ungetChar, 52
    ungetToken, 56
showImage hypertex, 491
showInput hypertex, 488
showLink hypertex, 486

showPage hypertex, 207
showPaste hypertex, 487
showPastebutton hypertex, 488
showScrollBars hypertex, 469
showSimpleBox hypertex, 489
showSpadcommand hypertex, 490
showText hypertex, 480
showTitleBar hypertex, 525
sigcldHandler hypertex, 162
sigusr2Handler hypertex, 162
spadbuf
    initParent, 100
    interpIO, 99
    main, 101
    spadbufFunctionChars, 98
    spadbufInterHandler, 97
spadbufFunctionChars spadbuf, 98
spadbufInterHandler spadbuf, 97
spadBusy shared, 68
spadErrorHandler shared, 67
splitpage hthits, 137
squirt hthits, 136
startFooter hypertex, 398
startNewline hypertex, 273
startScrolling hypertex, 397
startUserBuffer hypertex, 497
strCopy hypertex, 308
stringEqual shared, 43
stringHash shared, 43
strpostfix shared, 31
strPrefix ex2ht, 104
struct
    BeStruct, 31
    PgInfo, 130
switchFrames hypertex, 511

tempFileOpen shared, 37
textHeight hypertex, 284
textHeight1 hypertex, 285
textWidth hypertex, 276
toggleInputBox hypertex, 339
toggleRadioBox hypertex, 339
tokenName shared, 47
totalWidth hypertex, 281
toughEnter hypertex, 200
trailingSpace hypertex, 291
ttTopGroup hypertex, 299

unescapeString hypertex, 512
ungetChar shared, 52
ungetToken shared, 56
untexbuf hthits, 138
updateDB htadd, 121
updateInputsymbol hypertex, 180
uplink hypertex, 221

verbatimWidth hypertex, 275
verifyRecord hypertex, 303
void moveCursorForward hypertex, 183

widthOfDash hypertex, 275
windowCode hypertex, 400
windowEqual hypertex, 400
windowHeight hypertex, 295
windowId hypertex, 400
windowlinkHandler hypertex, 222
windowWidth hypertex, 295
wordWidth hypertex, 274
writable htadd, 116

Xvalue hypertex, 290