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.

Michael Albaugh	Cyril Alberga	Roy Adler
Christian Aistleitner	Richard Anderson	George Andrews
S.J. Atkins	Henry Baker	Martin Baker
Stephen Balzac	Yurij Baransky	David R. Barton
Thomas Baruchel	Gerald Baumgartner	Gilbert Baumslag
Michael Becker	Nelson H. F. Beebe	Jay Belanger
David Bindel	Fred Blair	Vladimir Bondarenko
Mark Botch	Raoul Bourquin	Alexandre Bouyer
Karen Braman	Peter A. Broadbery	Martin Brock
Manuel Bronstein	Stephen Buchwald	Florian Bundschuh
Luanne Burns	William Burge	Ralph Byers
Quentin Carpent	Robert Caviness	Bruce Char
Ondrej Certik	Tzu-Yi Chen	Cheekai Chin
David V. Chudnovsky	Gregory V. Chudnovsky	Mark Clements
James Cloos	Jia Zhao Cong	Josh Cohen
Christophe Conil	Don Coppersmith	George Corliss
Robert Corless	Gary Cornell	Meino Cramer
Jeremy Du Croz	David Cyganski	Nathaniel Daly
Timothy Daly Sr.	Timothy Daly Jr.	James H. Davenport
David Day	James Demmel	Didier Deshommes
Michael Dewar	Jack Dongarra	Jean Della Dora
Gabriel Dos Reis	Claire DiCrescendo	Sam Dooley
Lionel Ducos	Iain Duff	Lee Duhem
Martin Dunstan	Brian Dupee	Dominique Duval
Robert Edwards	Heow Eide-Goodman	Lars Erickson
Richard Fateman	Bertfried Fauser	Stuart Feldman
John Fletcher	Brian Ford	Albrecht Fortenbacher
George Frances	Constantine Frangos	Timothy Freeman
Korrinn Fu	Marc Gaetano	Rudiger Gebauer
Van de Geijn	Kathy Gerber	Patricia Gianni
Gustavo Goertkin	Samantha Goldrich	Holger Gollan
Teresa Gomez-Diaz	Laureano Gonzalez-Vega	Stephen Gortler
Johannes Grabmeier	Matt Grayson	Klaus Ebbe Grue
James Griesmer	Vladimir Grinberg	Oswald Gschnitzer
Ming Gu	Jocelyn Guidry	Gaetan Hache
Steve Hague	Satoshi Hamaguchi	Sven Hammarling
Mike Hansen	Richard Hanson	Richard Harke
Bill Hart	Vilya Harvey	Martin Hassner
Arthur S. Hathaway	Dan Hatton	Waldek Hebisch
Karl Hegbloom	Ralf Hemmecke	Henderson
Antoine Hersen	Roger House	Gernot Hueber
Pietro Iglio	Alejandro Jakubi	Richard Jenks
William Kahan	Kyriakos Kalorkoti	Kai Kaminski

Grant Keady
Ted Kosan
Bernhard Kutzler
Kaj Laurson
Frederic Lehobey
Ren-Cang Li
Richard Luczak
Alasdair McAndrew
Edi Meier
Victor S. Miller
H. Michael Moeller
Scott Morrison
William Naylor
John Nelder
Jinzhong Niu
Kostas Oikonomou
Bill Page
Michel Petitot
Frederick H. Pitts
Claude Quitte
Anatoly Raportirenko
Guilherme Reis
Jean Rivlin
Raymond Rogers
Philip Santas
Gerhard Schneider
Frithjof Schulze
V. Sima
Elena Smirnova
Christine Sundaresan
Eugene Surowitz
James Thatcher
Dylan Thurston
Themos T. Tsikas
Stephen Watt
M. Weller
Thorsten Werther
James T. Wheeler
Clifton J. Williamson
Robert Wisbauer
Knut Wolf
Clifford Yapp
Richard Zippel
Dan Zwillinger

Wilfrid Kendall
Paul Kosinski
Tim Lahey
George L. Legendre
Michel Levaud
Rudiger Loos
Camm Maguire
Bob McElrath
Ian Meikle
Gerard Milmeister
Michael Monagan
Joel Moses
Patrice Naudin
Godfrey Nolan
Michael O'Connor
Humberto Ortiz-Zuazaga
David Parnas
Didier Pinchon
Jose Alfredo Portes
Arthur C. Ralfs
Albert D. Rich
Huan Ren
Nicolas Robidoux
Michael Rothstein
Alfred Scheerhorn
Martin Schoenert
Fritz Schwarz
Nick Simicich
Jonathan Steinbach
Robert Sutor
Max Tegmark
Balbir Thomas
Steve Toleque
Gregory Vanuxem
Jaap Weel
Mark Wegman
Michael Wester
John M. Wiley
Stephen Wilson
Sandra Wityak
Yanyang Xiao
David Yun
Evelyn Zoernack

Tony Kennedy
Klaus Kusche
Larry Lambe
Franz Lehner
Howard Levy
Michael Lucks
Francois Maltey
Michael McGettrick
David Mentre
Mohammed Mobarak
Marc Moreno-Maza
Mark Murray
C. Andrew Neff
Arthur Norman
Summat Oemrawsingh
Julian A. Padget
Susan Pelzel
Ayal Pinkus
Gregorio Quintana-Orti
Norman Ramsey
Michael Richardson
Renaud Rioboo
Simon Robinson
Martin Rubey
William Schelter
Marshall Schor
Steven Segletes
William Sit
Fabio Stumbo
Moss E. Sweedler
T. Doug Telford
Mike Thomas
Barry Trager
Bernhard Wall
Juergen Weiss
James Wen
R. Clint Whaley
Berhard Will
Shmuel Winograd
Waldemar Wiwianka
Liu Xiaojun
Vadim Zhytnikov
Bruno Zuercher

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 Hypertex Call Graph **31**

4 include **87**
- 4.1 include/actions.h . . . 87
- 4.2 include/rgb.h . . . 90
- 4.3 include/spadcolors.h . . . 90
- 4.4 include/addfile.h1 . . . 91
- 4.5 include/all-hyper-proto.h1 . . . 91
- 4.6 include/bsdsignal.h . . . 92
- 4.7 include/bsdsignal.h1 . . . 93
- 4.8 include/com.h . . . 93
- 4.9 include/cond.h1 . . . 95
- 4.10 include/cursor.h1 . . . 95
- 4.11 include/debug.h . . . 95
- 4.12 include/dialog.h1 . . . 96
- 4.13 include/display.h1 . . . 97
- 4.14 include/edible.h . . . 97
- 4.15 include/edible.h1 . . . 101
- 4.16 include/edin.h1 . . . 101
- 4.17 include/event.h1 . . . 102
- 4.18 include/ex2ht.h1 . . . 103
- 4.19 include/extent1.h1 . . . 103

4.20 include/extent2.h1 . . . 104
4.21 include/fnct-key.h1 . . . 105
4.22 include/form-ext.h1 . . . 105
4.23 include/group.h1 . . . 106
4.24 include/halloc.h1 . . . 106
4.25 include/hash.h . . . 106
4.26 include/hash.h1 . . . 107
4.27 include/htadd.h1 . . . 107
4.28 include/hterror.h1 . . . 108
4.29 include/hthits.h1 . . . 108
4.30 include/htinp.h1 . . . 109
4.31 include/hyper.h1 . . . 109
4.32 include/initx.h1 . . . 110
4.33 include/input.h1 . . . 110
4.34 include/item.h1 . . . 111
4.35 include/keyin.h1 . . . 111
4.36 include/lex.h1 . . . 111
4.37 include/macro.h1 . . . 112
4.38 include/mem.h1 . . . 113
4.39 include/parse-aux.h1 . . . 113
4.40 include/parse.h1 . . . 114
4.41 include/parse-input.h1 . . . 115
4.42 include/parse-paste.h1 . . . 115
4.43 include/parse-types.h1 . . . 115
4.44 include/pixmap.h1 . . . 116
4.45 include/prt.h1 . . . 117
4.46 include/readbitmap.h1 . . . 117
4.47 include/scrollbar.h1 . . . 118
4.48 include/show-types.h1 . . . 118
4.49 include/sockio-c.h1 . . . 119
4.50 include/spadbuf.h1 . . . 120
4.51 include/spadcolors.h1 . . . 120
4.52 include/spadint.h1 . . . 121
4.53 include/titlebar.h1 . . . 122
4.54 include/util.h1 . . . 122
4.55 include/wct.h1 . . . 122

5 Shared Code **125**
BeStruct . . . 125
5.1 Shared Code for file handling . . . 125
strpostfix . . . 125
extendHT . . . 126
buildHtFilename . . . 126
pathname . . . 128
htFileOpen . . . 129
dbFileOpen . . . 129

tempFileOpen . . . 131
5.2 Shared Code for Hash Table Handling . . . 131
halloc . . . 131
hashInit . . . 132
freeHash . . . 132
hashInsert . . . 133
hashFind . . . 133
hashReplace . . . 133
hashDelete . . . 134
hashMap . . . 134
hashCopyEntry . . . 135
hashCopyTable . . . 135
stringHash . . . 135
stringEqual . . . 136
allocString . . . 136
5.3 Shared Code for Error Handling . . . 136
jump . . . 136
dumpToken . . . 137
printPageAndFilename . . . 137
printNextTenTokens . . . 138
printToken . . . 138
tokenName . . . 139
htperror . . . 140
5.4 Shared Code for Lexical Analyzer . . . 141
parserInit . . . 142
initScanner . . . 142
saveScannerState . . . 143
restoreScannerState . . . 143
ungetChar . . . 144
getChar . . . 144
getChar1 . . . 145
ungetToken . . . 147
getToken . . . 147
pushBeStack . . . 150
checkAndPopBeStack . . . 151
clearBeStack . . . 151
beType . . . 152
beginType . . . 153
endType . . . 154
keywordType . . . 155
getExpectedToken . . . 156
spadErrorHandler . . . 156
resetConnection . . . 157
spadBusy . . . 157
connectSpad . . . 158
5.5 htadd shared code . . . 158

5.6 hypertex shared code 162

6 Shared include files **167**
6.1 debug.c 167
6.2 include/hyper.h 167

7 The spadbuf function **179**
7.1 spadbuf Call Graph 179
7.2 Constants and Headers 180
System includes 180
Local includes 180
7.3 externs 181
7.4 local variables 181
7.5 Code 182
spadbufInterHandler 182
spadbufFunctionChars 182
interpIO 183
. 184
main 185

8 The ex2ht function **187**
8.1 ex2ht Call Graph 187
8.2 ex2ht Source Code 188
8.3 Constants and Headers 188
System includes 188
Local includes 189
8.4 defines 189
8.5 local variables 189
8.6 Code 189
allocString 189
strPrefix 190
getExTitle 190
exToHt 191
emitHeader 192
emitFooter 192
emitMenuEntry 192
emitSpadCommand 193
openCoverPage 193
closeCoverPage 194
closeCoverFile 194
emitCoverLink 194
addFile 195
main 195

9 The htadd command **197**
9.1 htadd Call Graph . . . 197
9.2 Constants and Headers . . . 202
System includes . . . 202
structs . . . 202
Local includes . . . 202
extern references . . . 203
defines . . . 203
forward declarations . . . 204
local variables . . . 204
9.3 The Shared Code . . . 205
9.4 Code . . . 205
parseArgs . . . 205
writable . . . 206
buildDBFilename . . . 206
addfile . . . 208
updateDB . . . 209
addNewPages . . . 210
copyFile . . . 211
getFilename . . . 212
deleteFile . . . 213
deleteDB . . . 213
main . . . 214

10 The hthits function **217**
10.1 hthits Call Graph . . . 217
10.2 Constants and Headers . . . 219
System includes . . . 219
defines . . . 219
structs . . . 219
Local includes . . . 220
local variables . . . 220
cmdline . . . 220
handleHtdb . . . 220
handleFile . . . 221
handleFilePages . . . 223
handlePage . . . 223
searchPage . . . 224
squirt . . . 225
splitpage . . . 225
untexbuf . . . 226
badDB . . . 227
regerr . . . 227
main . . . 227

11 The hypertex command **229**
11.1 Constants and Headers . . . 229
System includes . . . 229
11.2 structs . . . 230
Local includes . . . 230
11.3 structs . . . 230
11.4 defines . . . 231
11.5 externs . . . 235
11.6 local variables . . . 238
11.7 The Shared Code . . . 242
11.8 Code . . . 247
sigusr2Handler . . . 247
sigcldHandler . . . 247
cleanSocket . . . 247
initHash . . . 248
initPageStructs . . . 248
checkArguments . . . 248
makeServerConnections . . . 250
11.9 Condition Handling . . . 251
insertCond . . . 251
changeCond . . . 252
checkMemostack . . . 252
checkCondition . . . 253
11.10Dialog Handling . . . 254
redrawWin . . . 254
mystrncpy . . . 254
incLineNumbers . . . 254
decLineNumbers . . . 255
decreaseLineNumbers . . . 255
overwriteBuffer . . . 255
moveSymForward . . . 257
clearCursorline . . . 258
insertBuffer . . . 258
addBufferToSym . . . 260
drawInputsymbol . . . 261
updateInputsymbol . . . 262
drawCursor . . . 262
moveCursorHome . . . 263
moveCursorEnd . . . 264
void moveCursorForward . . . 264
moveCursorDown . . . 265
moveCursorUp . . . 265
clearCursor . . . 266
moveCursorBackward . . . 267
moveRestBack . . . 267
deleteRestOfLine . . . 268

backOverEoln . . . 269
moveBackOneChar . . . 271
backOverChar . . . 273
deleteEoln . . . 273
deleteOneChar . . . 275
deleteChar . . . 276
toughEnter . . . 276
enterNewLine . . . 278
dialog . . . 279
11.11 Format and Display a page . . . 282
showPage . . . 282
exposePage . . . 284
scrollPage . . . 285
pastePage . . . 286
11.12 Event Handling . . . 287
mainEventLoop . . . 287
handleEvent . . . 288
createWindow . . . 291
quitHyperDoc . . . 291
findPage . . . 292
downlink . . . 293
memolink . . . 293
killAxiomPage . . . 293
killPage . . . 294
returnlink . . . 294
uplink . . . 295
windowlinkHandler . . . 295
makeWindowLink . . . 295
lispwindowlinkHandler . . . 296
pasteButton . . . 296
helpForHyperDoc . . . 297
findButtonInList . . . 297
getHyperLink . . . 298
handleButton . . . 298
exitHyperDoc . . . 302
setWindow . . . 303
clearExposures . . . 304
getNewWindow . . . 304
setCursor . . . 307
changeCursor . . . 307
handleMotionEvent . . . 307
initCursorState . . . 308
initCursorStates . . . 308
makeBusyCursor . . . 308
makeBusyCursors . . . 309
HyperDocErrorHandler . . . 309

setErrorHandlers . . . 309
11.13Line Extent Computation . . . 310
computeInputExtent . . . 310
computePunctuationExtent . . . 310
computeWordExtent . . . 312
computeVerbatimExtent . . . 313
computeSpadsrctxtExtent . . . 313
computeDashExtent . . . 313
computeTextExtent . . . 314
computeBeginItemsExtent . . . 321
computeItemExtent . . . 322
computeMitemExtent . . . 322
endifExtent . . . 322
computeIfcondExtent . . . 323
computeCenterExtent . . . 324
computeBfExtent . . . 325
computeEmExtent . . . 325
computeItExtent . . . 325
computeRmExtent . . . 326
computeButtonExtent . . . 326
endbuttonExtent . . . 327
computePastebuttonExtent . . . 328
endpastebuttonExtent . . . 328
computePasteExtent . . . 329
computeSpadcommandExtent . . . 329
computeSpadsrcExtent . . . 330
endSpadcommandExtent . . . 330
endSpadsrcExtent . . . 331
computeMboxExtent . . . 332
computeBoxExtent . . . 332
computeIrExtent . . . 333
computeImageExtent . . . 334
computeTableExtent . . . 334
computeTitleExtent . . . 335
computeHeaderExtent . . . 336
computeFooterExtent . . . 337
computeScrollingExtent . . . 337
startNewline . . . 338
centerNodes . . . 338
punctuationWidth . . . 339
inputStringWidth . . . 339
wordWidth . . . 340
verbatimWidth . . . 340
widthOfDash . . . 340
textWidth . . . 341
totalWidth . . . 345

initExtents . . . 347
initTitleExtents . . . 347
initText . . . 348
textHeight . . . 348
textHeight1 . . . 348
maxX . . . 351
Xvalue . . . 353
trailingSpace . . . 354
insertBitmapFile . . . 354
insertPixmapFile . . . 355
plh . . . 356
11.14Handling forms . . . 356
computeFormPage . . . 357
windowWidth . . . 357
windowHeight . . . 357
formHeaderExtent . . . 358
formFooterExtent . . . 358
formScrollingExtent . . . 359
11.15Managing the HyperDoc group stack . . . 359
popGroupStack . . . 359
pushGroupStack . . . 360
initGroupStack . . . 360
emTopGroup . . . 361
rmTopGroup . . . 361
lineTopGroup . . . 361
bfTopGroup . . . 362
ttTopGroup . . . 362
pushActiveGroup . . . 362
pushSpadGroup . . . 363
initTopGroup . . . 363
centerTopGroup . . . 363
copyGroupStack . . . 364
freeGroupStack . . . 364
11.16Handle input, output, and Axiom communication . . . 365
makeRecord . . . 365
verifyRecord . . . 365
ht2Input . . . 366
makeInputFileName . . . 366
makePasteFileName . . . 367
makeTheInputFile . . . 367
makeInputFileFromPage . . . 368
strCopy . . . 369
inListAndNewer . . . 370
makeInputFileList . . . 371
printPasteLine . . . 371
getSpadOutput . . . 372

getGraphOutput . . . 372
sendCommand . . . 373
printPaste . . . 374
printGraphPaste . . . 374
11.17X Window window initialization code . . . 375
initializeWindowSystem . . . 375
initTopWindow . . . 377
openFormWindow . . . 378
initFormWindow . . . 379
setNameAndIcon . . . 380
getBorderProperties . . . 380
openWindow . . . 381
setSizeHints . . . 382
getGCs . . . 384
loadFont . . . 385
ingItColorsAndFonts . . . 385
changeText . . . 389
getColor . . . 389
mergeDatabases . . . 390
isIt850 . . . 392
11.18Handling user page interaction . . . 392
fillBox . . . 392
toggleInputBox . . . 393
toggleRadioBox . . . 393
clearRbs . . . 394
changeInputFocus . . . 394
nextInputFocus . . . 395
prevInputFocus . . . 395
returnItem . . . 396
deleteItem . . . 396
11.19Manipulate the item stack . . . 397
pushItemStack . . . 397
clearItemStack . . . 397
popItemStack . . . 398
copyItemStack . . . 398
freeItemStack . . . 399
11.20Keyboard handling . . . 399
handleKey . . . 399
getModifierMask . . . 402
initKeyin . . . 403
11.21Handle page macros . . . 404
scanHyperDoc . . . 404
number . . . 405
loadMacro . . . 405
initParameterElem . . . 407
pushParameters . . . 407

popParameters . . . 408
parseMacro . . . 408
getParameterStrings . . . 409
parseParameters . . . 411
11.22Memory management routines . . . 412
freeIfNonNULL . . . 412
allocHdWindow . . . 412
freeHdWindow . . . 413
allocNode . . . 413
freeNode . . . 414
allocIfnode . . . 417
allocCondnode . . . 418
freeCond . . . 418
allocPage . . . 418
freePage . . . 419
freePaste . . . 420
freePastebutton . . . 421
freePastearea . . . 421
freeString . . . 422
freeDepend . . . 422
dontFree . . . 422
freeLines . . . 423
freeInputItem . . . 423
freeInputList . . . 423
freeInputBox . . . 424
freeRadioBoxes . . . 424
allocInputline . . . 424
allocPasteNode . . . 425
allocPatchstore . . . 425
freePatch . . . 426
allocInputbox . . . 426
allocRbs . . . 426
allocButtonList . . . 427
freeButtonList . . . 427
resizeBuffer . . . 427
11.23Page parsing routines . . . 428
PushMR . . . 428
PopMR . . . 428
loadPage . . . 429
displayPage . . . 429
formatPage . . . 430
parseFromString . . . 431
parseTitle . . . 431
parseHeader . . . 432
initParsePage . . . 432
initParsePatch . . . 433

parsePage . . . 433
parseHyperDoc . . . 434
parsePageFromSocket . . . 441
parsePageFromUnixfd . . . 442
startScrolling . . . 443
startFooter . . . 443
endAPage . . . 444
parseReplacepage . . . 445
windowEqual . . . 445
windowCode . . . 445
windowId . . . 445
readHtDb . . . 446
readHtFile . . . 447
makeLinkWindow . . . 450
makePasteWindow . . . 452
makeSpecialPage . . . 452
main . . . 453
addDependencies . . . 453
isNumber . . . 454
parserError . . . 455
getFilename . . . 455
getInputString . . . 456
getWhere . . . 457
findFp . . . 457
11.24Handle InputString, SimpleBox, RadioBox input . . . 458
makeInputWindow . . . 458
makeBoxWindow . . . 459
initializeDefault . . . 459
parseInputstring . . . 460
parseSimplebox . . . 462
parseRadiobox . . . 463
addBoxToRbList . . . 465
checkOthers . . . 466
insertItem . . . 466
initPasteItem . . . 467
repasteItem . . . 467
currentItem . . . 468
alreadyThere . . . 468
parseRadioboxes . . . 469
11.25Routines for paste-in areas . . . 470
parsePaste . . . 470
parsePastebutton . . . 472
parsePatch . . . 473
loadPatch . . . 475
11.26parsing routines for node types . . . 476
parseIfcond . . . 476

parseCondnode . . . 478
parseHasreturnto . . . 479
parseNewcond . . . 479
parseSetcond . . . 479
parseBeginItems . . . 480
parseItem . . . 481
parseMitem . . . 481
parseVerbatim . . . 482
parseInputPix . . . 483
parseCenterline . . . 484
parseCommand . . . 484
parseButton . . . 485
parseSpadcommand . . . 486
parseSpadsrc . . . 487
parseEnv . . . 487
parseValue1 . . . 488
parseValue2 . . . 489
parseTable . . . 489
parseBox . . . 490
parseMbox . . . 491
parseFree . . . 491
parseHelp . . . 492
11.27Reading bitmaps . . . 492
HTReadBitmapFile . . . 492
readHot . . . 495
readWandH . . . 495
insertImageStruct . . . 496
11.28Scrollbar handling routines . . . 496
makeScrollBarWindows . . . 497
drawScroller3DEffects . . . 499
showScrollBars . . . 500
moveScroller . . . 501
drawScrollLines . . . 501
calculateScrollBarMeasures . . . 502
linkScrollBars . . . 503
scrollUp . . . 504
scrollUpPage . . . 505
scrollToFirstPage . . . 505
scrollDown . . . 505
scrollDownPage . . . 506
scrollScroller . . . 506
hideScrollBars . . . 507
getScrollBarMinimumSize . . . 508
ch . . . 508
changeWindowBackgroundPixmap . . . 508
11.29Display text object . . . 509

showText . . . 509
showLink . . . 514
showPaste . . . 515
showPastebutton . . . 516
showInput . . . 516
showSimpleBox . . . 517
showSpadcommand . . . 517
showImage . . . 518
11.30 Axiom communication interface . . . 519
issueSpadcommand . . . 519
sendPile . . . 520
issueDependentCommands . . . 521
markAsExecuted . . . 522
startUserBuffer . . . 522
clearExecutionMarks . . . 523
acceptMenuConnection . . . 524
acceptMenuServerConnection . . . 525
printToString . . . 526
printToString1 . . . 526
issueServerCommand . . . 531
issueServerpaste . . . 532
issueUnixcommand . . . 533
issueUnixlink . . . 533
issueUnixpaste . . . 534
serviceSessionSocket . . . 534
switchFrames . . . 535
sendLispCommand . . . 535
escapeString . . . 535
unescapeString . . . 536
closeClient . . . 536
printSourceToString . . . 537
printSourceToString1 . . . 537
11.31 Produce titlebar . . . 545
makeTitleBarWindows . . . 545
showTitleBar . . . 546
linkTitleBarWindows . . . 547
readTitleBarImages . . . 548
getTitleBarMinimumSize . . . 549
main . . . 549

12 The htsearch script **553**

13 The presea script **555**
13.1 token.h . . . 556

14 The Bitmaps **561**
14.1 ht_icon . . . 561
14.2 exit.bitmap . . . 562
14.3 help2.bitmap . . . 562
14.4 return3.bitmap . . . 563
14.5 up3.bitmap . . . 564
14.6 noop.bitmap . . . 564
14.7 exit3d.bitmap . . . 565
14.8 help3d.bitmap . . . 566
14.9 home3d.bitmap . . . 566
14.10 up3d.bitmap . . . 567
14.11 noop3d.bitmap . . . 568

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))

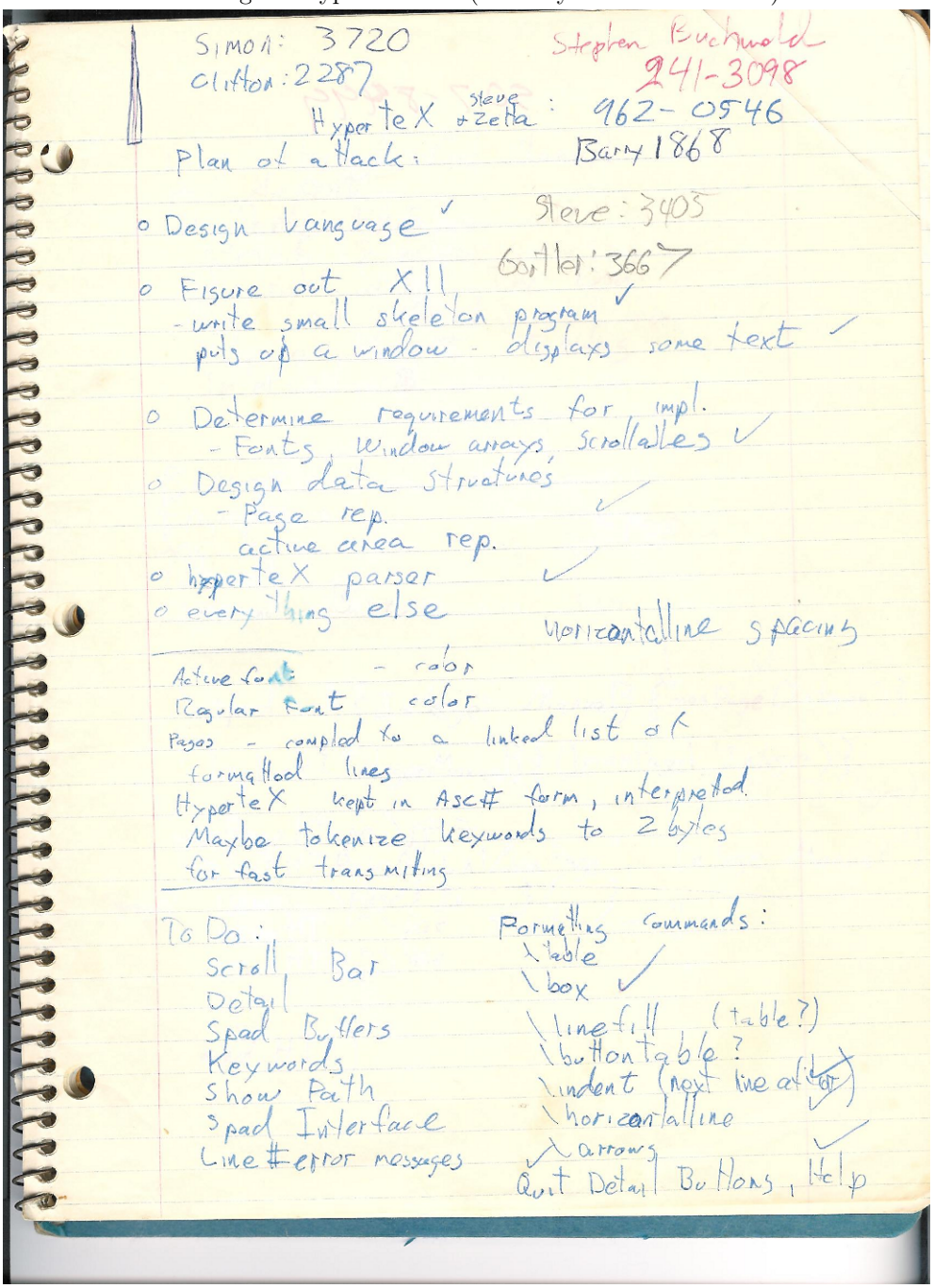
Chapter 1

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)



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 5.4 on page 142
- the XENVIRONMENT shell variable, if set is used to find the X database to merge, otherwise it uses .Xdefaults from the HOME directory. See 11.17 on page 390
- NOFREE shell variable is supposed to turn off freeing memory. See 11.22 on page 419
- SPADNUM shell variable is the number of the spad communication socket. See 11.30 on page 520

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}{(|cO2aff|)}
\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

Hypertex Call Graph

This was generated by the GNU cflow program with the argument list. Note that the line:NNNN numbers refer to the line in the code after it has been tangled from this file.

```
cflow --emacs -l -n -b -T --omit-arguments hypertex.c
```

```
;; This file is generated by GNU cflow 1.3. -*- cflow -*-
    2 {   0} +-main() <int main () line:14475>
    3 {   1}   +-checkArguments() <void checkArguments () line:3041>
    4 {   2}   | +-fprintf()
    5 {   2}   | \-exit()
    6 {   1}   +-initHash() <void initHash () line:3014>
    7 {   2}   | +-hashInit() <void hashInit () line:2091>
    8 {   3}   |   \-halloc() <char *halloc () line:2070>
    9 {   4}   |     +-fopen()
   10 {   4}   |     +-malloc()
   11 {   4}   |     +-fprintf()
   12 {   4}   |     +-sprintf()
   13 {   4}   |     \-exit()
   14 {   2}   | +-stringEqual() <int stringEqual () line:2185>
   15 {   3}   |   \-strcmp()
   16 {   2}   | +-stringHash() <int stringHash () line:2177>
   17 {   2}   | +-windowEqual() <int windowEqual () line:10409>
   18 {   2}   | \-windowCode() <int windowCode () line:10413>
   19 {   1}   +-parserInit() <void parserInit () line:2326>
   20 {   2}   | +-hashInit() <void hashInit () line:2091> [see 7]
   21 {   2}   | +-stringEqual() <int stringEqual () line:2185> [see 14]
   22 {   2}   | +-stringHash() <int stringHash () line:2177>
   23 {   2}   | +-halloc() <char *halloc () line:2070> [see 8]
   24 {   2}   | \-hashInsert() <void hashInsert () line:2104>
   25 {   3}   |   +-halloc() <char *halloc () line:2070> [see 8]
   26 {   3}   |   \-fprintf()
   27 {   1}   +-readHtDb() <void readHtDb () line:10428>
```

```
28 {   2}     +-hashInit() <void hashInit () line:2091> [see 7]
29 {   2}     +-stringEqual() <int stringEqual () line:2185> [see 14]
30 {   2}     +-stringHash() <int stringHash () line:2177>
31 {   2}     +-dbFileOpen() <FILE *dbFileOpen () line:2929>
32 {   3}     | +-getenv()
33 {   3}     | +-fprintf()
34 {   3}     | +-exit()
35 {   3}     | +-halloc() <char *halloc () line:2070> [see 8]
36 {   3}     | +-strcpy()
37 {   3}     | +-strcat()
38 {   3}     | \-fopen()
39 {   2}     +-readHtFile() <void readHtFile () line:10468>
40 {   3}     | +-initScanner() <void initScanner () line:2343>
41 {   4}     | | +-getenv()
42 {   4}     | | \-strcmp()
43 {   3}     | +-strlen()
44 {   3}     | +-getc()
45 {   3}     | +-getFilename() <int getFilename () line:10795>
46 {   4}     | | +-getChar() <int getChar () line:2490>
47 {   5}     | | | \-getChar1() <int getChar1 () line:2433>
48 {   6}     | | |   +-getc()
49 {   6}     | | |   +-get_int()
50 {   6}     | | |   +-spadErrorHandler()
               | | |   | <void spadErrorHandler () line:1864>
51 {   7}     | | |   | +-longjmp()
52 {   7}     | | |   | +-fprintf()
53 {   7}     | | |   | \-exit()
54 {   6}     | | |   +-get_string_buf()
55 {   6}     | | |   \-fprintf()
56 {   4}     | | +-whitespace()
57 {   4}     | | +-fprintf()
58 {   4}     | | +-exit()
59 {   4}     | | +-filedelim()
60 {   4}     | | \-ungetChar() <void ungetChar () line:2400>
61 {   3}     | +-allocString() <char *allocString () line:2189>
62 {   4}     |   +-halloc() <char *halloc () line:2070> [see 8]
63 {   4}     |   +-strlen()
64 {   4}     |   \-strcpy()
65 {   3}     | +-strcpy()
66 {   3}     | +-strcat()
67 {   3}     | +-free()
68 {   3}     | +-hashFind() <char *hashFind () line:2139>
69 {   3}     | +-hashInsert() <void hashInsert () line:2104> [see 24]
70 {   3}     | +-stat()
71 {   3}     | +-sprintf()
72 {   3}     | +-perror()
73 {   3}     | +-exit()
74 {   3}     | +-getToken() <int getToken () line:2535> (R)
75 {   4}     | | +-strcpy()
76 {   4}     | | +-free()
```

```
 77 {   4}     | | +-getChar() <int getChar () line:2490> [see 46]
 78 {   4}     | | +-whitespace()
 79 {   4}     | | +-ungetChar() <void ungetChar () line:2400> [see 60]
 80 {   4}     | | +-getToken() <int getToken () line:2535>
               | | |(recursive: see 74) [see 74]
 81 {   4}     | | +-isalpha()
 82 {   4}     | | +-keywordType() <int keywordType () line:2865> (R)
 83 {   5}     | |   +-hashFind() <char *hashFind () line:2139> [see 68]
 84 {   5}     | |   +-beginType() <int beginType () line:2803> (R)
 85 {   6}     | |     +-beType() <int beType () line:2735> (R)
 86 {   7}     | |       +-getExpectedToken()
               | |       | <void getExpectedToken () line:2406> (R)
 87 {   8}     | |         +-getToken()
               | |         | <int getToken () line:2535>
               | |         | (recursive: see 74) [see 74]
 88 {   8}     | |         +-tokenName() <void tokenName () line:2204>
 89 {   9}     | |         | +-strcpy()
 90 {   9}     | |         | \-sprintf()
 91 {   8}     | |         +-fprintf()
 92 {   8}     | |         +-printPageAndFilename()
               | |         | <void printPageAndFilename () line:2286>
 93 {   9}     | |         | +-sprintf()
 94 {   9}     | |         | \-fprintf()
 95 {   8}     | |         +-printNextTenTokens()
               | |         | <void printNextTenTokens () line:2313> (R)
 96 {   9}     | |         | +-fprintf()
 97 {   9}     | |         | +-getToken()
               | |         | | <int getToken () line:2535>
               | |         | | (recursive: see 74) [see 74]
 98 {   9}     | |         | \-printToken() <void printToken () line:2276>
 99 {  10}     | |         |   +-printf()
100 {  10}     | |         |   +-tokenName()
               | |         |   <void tokenName () line:2204> [see 88]
101 {  10}     | |         |   \-fflush()
102 {   8}     | |         +-longjmp()
103 {   8}     | |         \-exit()
104 {   7}     | |       \-strcmp()
105 {   6}     | |     +-fprintf()
106 {   6}     | |     +-printPageAndFilename()
               | |     | <void printPageAndFilename () line:2286> [see 92]
107 {   6}     | |     +-printNextTenTokens()
               | |     | <void printNextTenTokens () line:2313> (R) [see 95]
108 {   6}     | |     +-jump() <void jump () line:2196>
109 {   7}     | |       +-exit()
110 {   7}     | |       +-longjmp()
111 {   7}     | |       \-fprintf()
112 {   6}     | |     \-pushBeStack() <void pushBeStack () line:2689>
113 {   7}     | |       +-halloc() <char *halloc () line:2070> [see 8]
114 {   7}     | |       \-allocString()
               | |         <char *allocString () line:2189> [see 61]
```

```
115 {   5}     | |   \-endType() <int endType () line:2828> (R)
116 {   6}     | |     +-beType() <int beType () line:2735> (R) [see 85]
117 {   6}     | |     +-fprintf()
118 {   6}     | |     +-printPageAndFilename()
                | |     | <void printPageAndFilename () line:2286> [see 92]
119 {   6}     | |     +-printNextTenTokens()
                | |     | <void printNextTenTokens () line:2313> (R) [see 95]
120 {   6}     | |     +-jump() <void jump () line:2196> [see 108]
121 {   6}     | |     \-checkAndPopBeStack()
                | |       <void checkAndPopBeStack () line:2711> (R)
122 {   7}     | |       +-fprintf()
123 {   7}     | |       +-printPageAndFilename()
                | |       | <void printPageAndFilename () line:2286> [see 92]
124 {   7}     | |       +-printNextTenTokens()
                | |       | <void printNextTenTokens () line:2313>
                | |       | (R) [see 95]
125 {   7}     | |       +-jump() <void jump () line:2196> [see 108]
126 {   7}     | |       \-free()
127 {   4}     | | +-isdigit()
128 {   4}     | | \-delim()
129 {   3}     | +-atoi()
130 {   3}     | +-fprintf()
131 {   3}     | +-ungetc()
132 {   3}     | +-halloc() <char *halloc () line:2070> [see 8]
133 {   3}     | +-strcmp()
134 {   3}     | +-allocPatchstore()
                | | <PatchStore *allocPatchstore () line:9668>
135 {   4}     | | \-halloc() <char *halloc () line:2070> [see 8]
136 {   3}     | \-freePatch() <void freePatch () line:9675>
137 {   4}     |   \-free()
138 {   2}     +-fclose()
139 {   2}     +-fprintf()
140 {   2}     +-exit()
141 {   2}     +-freeHash() <void freeHash () line:2160>
142 {   3}     |   \-free()
143 {   2}     \-freeString() <void freeString () line:9589>
144 {   3}       \-freeIfNonNULL() <void freeIfNonNULL () line:9201>
145 {   4}         \-free()
146 {   1}   +-initializeWindowSystem()
              | <void initializeWindowSystem () line:7819>
147 {   2}   | +-XOpenDisplay()
148 {   2}   | +-fprintf()
149 {   2}   | +-exit()
150 {   2}   | +-DefaultScreen()
151 {   2}   | +-XGContextFromGC()
152 {   2}   | +-DefaultGC()
153 {   2}   | +-DefaultColormap()
154 {   2}   | +-WhitePixel()
155 {   2}   | +-XQueryColor()
156 {   2}   | +-BlackPixel()
```

```
157 {   2}   | +-XCreateBitmapFromData()
158 {   2}   | +-RootWindow()
159 {   2}   | +-XCreatePixmapCursor()
160 {   2}   | +-XCreateFontCursor()
161 {   2}   | +-ingItColorsAndFonts()
             | | <void ingItColorsAndFonts () line:8206>
162 {   3}   | | +-DefaultColormap()
163 {   3}   | | +-initGroupStack() <void initGroupStack () line:7325>
164 {   4}   | | | \-halloc() <char *halloc () line:2070> [see 8]
165 {   3}   | | +-mergeDatabases() <void mergeDatabases () line:8427>
166 {   4}   | | | +-XrmInitialize()
167 {   4}   | | | +-strcpy()
168 {   4}   | | | +-strcat()
169 {   4}   | | | +-XrmGetFileDatabase()
170 {   4}   | | | +-XrmMergeDatabases()
171 {   4}   | | | +-XResourceManagerString()
172 {   4}   | | | +-XrmGetStringDatabase()
173 {   4}   | | | +-getenv()
174 {   4}   | | | +-strlen()
175 {   4}   | | | \-gethostname()
176 {   3}   | | +-XrmGetResource()
177 {   3}   | | +-strncpy()
178 {   3}   | | +-strcpy()
179 {   3}   | | +-loadFont() <void loadFont () line:8193>
180 {   4}   | |   +-XLoadQueryFont()
181 {   4}   | |   +-fprintf()
182 {   4}   | |   +-XQueryFont()
183 {   4}   | |   +-XGContextFromGC()
184 {   4}   | |   +-DefaultGC()
185 {   4}   | |   \-exit()
186 {   3}   | | +-isIt850() <int isIt850 () line:8470>
187 {   4}   | |   +-XInternAtom()
188 {   4}   | |   +-XGetAtomName()
189 {   4}   | |   +-strcmp()
190 {   4}   | |   \-XFree()
191 {   3}   | | +-DisplayPlanes()
192 {   3}   | | +-BlackPixel()
193 {   3}   | | +-WhitePixel()
194 {   3}   | | +-getColor() <int getColor () line:8384>
195 {   4}   | | | +-printf()
196 {   4}   | | | +-strcpy()
197 {   4}   | | | +-strcat()
198 {   4}   | | | +-XrmGetResource()
199 {   4}   | | | +-strncpy()
200 {   4}   | | | +-XAllocNamedColor()
201 {   4}   | | | \-fprintf()
202 {   3}   | | \-makeColors()
203 {   2}   | \-initText() <void initText () line:6865>
204 {   1}   +-initKeyin() <void initKeyin () line:8876>
205 {   2}   | +-getModifierMask()
```

```
             | \ <unsigned int getModifierMask () line:8852>
206 {   3}   |   +-XGetModifierMapping()
207 {   3}   |   +-XKeysymToKeycode()
208 {   3}   |   \-XFreeModifiermap()
209 {   2}   | +-XTextWidth()
210 {   2}   | +-XGetDefault()
211 {   2}   | +-halloc() <char *halloc () line:2070> [see 8]
212 {   2}   | +-strlen()
213 {   2}   | \-strcpy()
214 {   1}   +-initTopWindow() <int initTopWindow () line:7871>
215 {   2}     +-allocHdWindow() <HDWindow *allocHdWindow () line:9207>
216 {   3}       +-halloc() <char *halloc () line:2070> [see 8]
217 {   3}       +-initPageStructs() <void initPageStructs () line:3029>
218 {   3}       +-hashInit() <void hashInit () line:2091> [see 7]
219 {   3}       +-stringEqual() <int stringEqual () line:2185> [see 14]
220 {   3}       +-stringHash() <int stringHash () line:2177>
221 {   3}       +-hashCopyTable() <HashTable *hashCopyTable () line:2915>
222 {   4}         +-halloc() <char *halloc () line:2070> [see 8]
223 {   4}         \-hashCopyEntry()
                     <HashEntry *hashCopyEntry () line:2903> (R)
224 {   5}           +-halloc() <char *halloc () line:2070> [see 8]
225 {   5}           \-hashCopyEntry()
                       <HashEntry *hashCopyEntry () line:2903>
                       (recursive: see 223) [see 223]
226 {   3}       \-makeSpecialPages() <void makeSpecialPages () line:10720>
227 {   4}         +-hashInsert() <void hashInsert () line:2104> [see 24]
228 {   4}         \-makeSpecialPage()
                     <HyperDocPage *makeSpecialPage () line:10708>
229 {   5}           +-allocPage() <HyperDocPage *allocPage () line:9472>
230 {   6}             +-halloc() <char *halloc () line:2070> [see 8]
231 {   6}             \-allocString()
                         <char *allocString () line:2189> [see 61]
232 {   5}           +-fprintf()
233 {   5}           +-exit()
234 {   5}           \-free()
235 {   2}     +-allocPage() <HyperDocPage *allocPage () line:9472> [see 229]
236 {   2}     +-hashFind() <char *hashFind () line:2139> [see 68]
237 {   2}     +-fprintf()
238 {   2}     +-exit()
239 {   2}     +-openWindow() <void openWindow () line:8050>
240 {   3}     | +-strcpy()
241 {   3}     | +-XrmGetResource()
242 {   3}     | +-strncpy()
243 {   3}     | +-XGeometry()
244 {   3}     | +-getBorderProperties()
             | | <int getBorderProperties () line:8023>
245 {   4}     | | +-atoi()
246 {   4}     | | +-fprintf()
247 {   4}     | | +-DisplayPlanes()
248 {   4}     | | +-BlackPixel()
```

```
249 {   4}     | | +-DefaultColormap()
250 {   4}     | | \-getColor() <int getColor () line:8384> [see 194]
251 {   3}     | +-XCreateSimpleWindow()
252 {   3}     | +-RootWindow()
253 {   3}     | +-WhitePixel()
254 {   3}     | +-makeScrollBarWindows()
                | | <void makeScrollBarWindows () line:12390>
255 {   4}     | | +-fprintf()
256 {   4}     | | +-exit()
257 {   4}     | | +-XCreatePixmapFromBitmapData()
258 {   4}     | | +-RootWindow()
259 {   4}     | | +-DefaultDepth()
260 {   4}     | | +-XCreateSimpleWindow()
261 {   4}     | | \-XChangeWindowAttributes()
262 {   3}     | +-makeTitleBarWindows()
                | | <void makeTitleBarWindows () line:14315>
263 {   4}     | | +-readTitleBarImages()
                | |   <void readTitleBarImages () line:14433>
264 {   5}     | |   +-getenv()
265 {   5}     | |   +-sprintf()
266 {   5}     | |   \-HTReadBitmapFile()
                | |     <XImage *HTReadBitmapFile () line:12233>
267 {   6}     | |     +-XCreateImage()
268 {   6}     | |     +-DefaultVisual()
269 {   6}     | |     +-zzopen()
270 {   6}     | |     +-fprintf()
271 {   6}     | |     +-exit()
272 {   6}     | |     +-readWandH() <int readWandH () line:12343>
273 {   7}     | |     | +-fgets()
274 {   7}     | |     | \-sscanf()
275 {   6}     | |     +-fgets()
276 {   6}     | |     +-readHot() <int readHot () line:12327>
277 {   7}     | |     | +-sscanf()
278 {   7}     | |     | \-fgets()
279 {   6}     | |     +-sscanf()
280 {   6}     | |     +-halloc() <char *halloc () line:2070> [see 8]
281 {   6}     | |     +-fscanf()
282 {   6}     | |     \-fclose()
283 {   4}     | | +-XCreateSimpleWindow()
284 {   4}     | | \-XChangeWindowAttributes()
285 {   3}     | +-setNameAndIcon() <void setNameAndIcon () line:7996>
286 {   4}     | | +-ch() <int ch () line:12776>
287 {   4}     | | +-strlen()
288 {   4}     | | +-XSetClassHint()
289 {   4}     | | +-XStoreName()
290 {   4}     | | +-XCreateBitmapFromData()
291 {   4}     | | +-XSetWMHints()
292 {   4}     | | \-XSetIconName()
293 {   3}     | +-setSizeHints() <void setSizeHints () line:8091>
294 {   4}     | | +-strcpy()
```

```
295 {   4}     | | +-XGetGeometry()
296 {   4}     | | +-getWindowPositionXY()
297 {   4}     | | +-fprintf()
298 {   4}     | | +-XrmGetResource()
299 {   4}     | | +-strncpy()
300 {   4}     | | +-XParseGeometry()
301 {   4}     | | +-XGeometry()
302 {   4}     | | +-getTitleBarMinimumSize()
                | |   <void getTitleBarMinimumSize () line:14470>
303 {   4}     | | +-XSetNormalHints()
304 {   4}     | | \-XFlush()
305 {   3}     | +-XSelectInput()
306 {   3}     | \-XDefineCursor()
307 {   2}     +-getGCs() <void getGCs () line:8161>
308 {   3}     | +-XCreateGC()
309 {   3}     | +-XSetLineAttributes()
310 {   3}     | +-XCreateBitmapFromData()
311 {   3}     | +-RootWindow()
312 {   3}     | +-XSetFont()
313 {   3}     | +-XSetBackground()
314 {   3}     | \-XSetForeground()
315 {   2}     +-XMapWindow()
316 {   2}     +-hashInsert() <void hashInsert () line:2104> [see 24]
317 {   2}     +-changeText() <void changeText () line:8372>
318 {   3}       +-XChangeGC()
319 {   3}       \-XSetFont()
320 {   2}     \-XChangeWindowAttributes()
321 {   1}   +-fprintf()
322 {   1}   +-exit()
323 {   1}   +-bsdSignal()
324 {   1}   +-sigusr2Handler() <void sigusr2Handler () line:2998>
325 {   1}   +-sigcldHandler() <void sigcldHandler () line:3003>
326 {   2}     \-wait()
327 {   1}   +-makeServerConnections()
              | <void makeServerConnections () line:3089>
328 {   2}   | +-open_server()
329 {   2}   | +-fprintf()
330 {   2}   | +-atexit()
331 {   2}   | +-cleanSocket() <void cleanSocket () line:3008>
332 {   3}   |   +-make_server_name()
333 {   3}   |   \-unlink()
334 {   2}   | +-connect_to_local_server()
335 {   2}   | \-exit()
336 {   1}   +-ht2Input() <void ht2Input () line:7475>
337 {   2}   | +-bsdSignal()
338 {   2}   | +-allocHdWindow()
              | | <HDWindow *allocHdWindow () line:9207> [see 215]
339 {   2}   | +-initGroupStack()
              | | <void initGroupStack () line:7325> [see 163]
340 {   2}   | +-makeInputFileList() <void makeInputFileList () line:7670>
```

```
341 {   3}   |   +-makeInputFileName() <char *makeInputFileName () line:7494>
342 {   4}   |     +-strcpy()
343 {   4}   |     \-strlen()
344 {   3}   |   +-halloc() <char *halloc () line:2070> [see 8]
345 {   3}   |   +-strlen()
346 {   3}   |   \-strcpy()
347 {   2}   | +-makeTheInputFile() <void makeTheInputFile () line:7516>
348 {   3}   |   +-makeInputFileName()
              |   | <char *makeInputFileName () line:7494> [see 341]
349 {   3}   |   +-inListAndNewer() <int inListAndNewer () line:7618>
350 {   4}   |   | +-strcmp()
351 {   4}   |   | +-strCopy() <char *strCopy () line:7612>
352 {   5}   |   | | +-halloc() <char *halloc () line:2070> [see 8]
353 {   5}   |   | | +-strlen()
354 {   5}   |   | | \-strcpy()
355 {   4}   |   | +-stat()
356 {   4}   |   | +-printf()
357 {   4}   |   | \-unlink()
358 {   3}   |   +-printf()
359 {   3}   |   +-setjmp()
360 {   3}   |   +-loadPage() <void loadPage () line:9758>
361 {   4}   |   | +-initScanner() <void initScanner () line:2343> [see 40]
362 {   4}   |   | \-formatPage() <HyperDocPage *formatPage () line:9805>
363 {   5}   |   |   +-allocPage()
              |   |   | <HyperDocPage *allocPage () line:9472> [see 229]
364 {   5}   |   |   +-hashReplace() <char *hashReplace () line:2148>
365 {   5}   |   |   +-findFp() <FILE *findFp () line:10878>
366 {   6}   |   |   | +-hashFind() <char *hashFind () line:2139> [see 68]
367 {   6}   |   |   | +-htFileOpen() <FILE *htFileOpen () line:2041>
368 {   7}   |   |   | | +-buildHtFilename()
              |   |   | | | <int buildHtFilename () line:1946>
369 {   8}   |   |   | | | +-cwd()
370 {   8}   |   |   | | | +-getcwd()
371 {   8}   |   |   | | | +-strcpy()
372 {   8}   |   |   | | | +-strcat()
373 {   8}   |   |   | | | +-strlen()
374 {   8}   |   |   | | | +-fprintf()
375 {   8}   |   |   | | | +-exit()
376 {   8}   |   |   | | | +-extendHT() <void extendHT () line:1938>
377 {   9}   |   |   | | | | +-strpostfix() <int strpostfix () line:1928>
378 {  10}   |   |   | | | | | \-strlen()
379 {   9}   |   |   | | | | \-strcat()
380 {   8}   |   |   | | | +-access()
381 {   8}   |   |   | | | +-pathname() <int pathname () line:1913>
382 {   8}   |   |   | | | +-getenv()
383 {   8}   |   |   | | | +-halloc() <char *halloc () line:2070> [see 8]
384 {   8}   |   |   | | | \-strcmp()
385 {   7}   |   |   | | +-fprintf()
386 {   7}   |   |   | | +-exit()
387 {   7}   |   |   | | +-fopen()
```

```
388 {   7}   |   |   | | \-perror()
389 {   6}   |   |   | +-hashInsert() <void hashInsert () line:2104> [see 24]
390 {   6}   |   |   | +-fseek()
391 {   6}   |   |   | +-perror()
392 {   6}   |   |   | \-longjmp()
393 {   5}   |   |   +-allocString()
              |   |   | <char *allocString () line:2189> [see 61]
394 {   5}   |   |   \-parsePage() <void parsePage () line:9917>
395 {   6}   |   |     +-initParsePage() <void initParsePage () line:9882>
396 {   7}   |   |     | +-freeInputList() <void freeInputList () line:9620>
397 {   8}   |   |     |   +-freeInputItem()
              |   |     |   | <void freeInputItem () line:9613>
398 {   9}   |   |     |   | +-freeIfNonNULL()
              |   |     |   | | <void freeIfNonNULL () line:9201> [see 144]
399 {   9}   |   |     |   | +-freeLines() <void freeLines () line:9601>
400 {  10}   |   |     |   | | \-free()
401 {   9}   |   |     |   | \-XDestroyWindow()
402 {   8}   |   |     |   \-free()
403 {   7}   |   |     | +-initTopGroup() <void initTopGroup () line:7392>
404 {   8}   |   |     | | +-popGroupStack() <int popGroupStack () line:7294>
405 {   9}   |   |     | |   +-free()
406 {   9}   |   |     | |   \-changeText() <void changeText () line:8372>
              |   |     | |     [see 317]
407 {   8}   |   |     | | \-changeText() <void changeText () line:8372>
              |   |     | |   [see 317]
408 {   7}   |   |     | +-clearBeStack() <int clearBeStack () line:2700>
409 {   8}   |   |     |   \-free()
410 {   7}   |   |     | +-hashInit() <void hashInit () line:2091> [see 7]
411 {   7}   |   |     | +-windowEqual() <int windowEqual () line:10409>
412 {   7}   |   |     | \-windowCode() <int windowCode () line:10413>
413 {   6}   |   |     +-getExpectedToken()
              |   |     | <void getExpectedToken () line:2406> (R) [see 86]
414 {   6}   |   |     +-allocString()
              |   |     | <char *allocString () line:2189> [see 61]
415 {   6}   |   |     +-parseTitle() <void parseTitle () line:9833>
416 {   7}   |   |     | +-PushMR() <void PushMR () line:9734>
417 {   8}   |   |     |   \-halloc() <char *halloc () line:2070> [see 8]
418 {   7}   |   |     | +-getExpectedToken()
              |   |     |   <void getExpectedToken () line:2406> (R) [see 86]
419 {   7}   |   |     | +-allocNode() <TextNode *allocNode () line:9265>
420 {   8}   |   |     |   \-halloc() <char *halloc () line:2070> [see 8]
421 {   7}   |   |     | +-parseHyperDoc()
              |   |     |   <void parseHyperDoc () line:9938> (R)
422 {   8}   |   |     |   +-getToken() <int getToken () line:2535>
              |   |     |   | (R) [see 74]
423 {   8}   |   |     |   +-parseSpadsrc()
              |   |     |     <void parseSpadsrc () line:12034> (R)
424 {   9}   |   |     |     +-allocNode() <TextNode *allocNode () line:9265>
              |   |     |     | [see 419]
425 {   9}   |   |     |     +-getChar() <int getChar () line:2490> [see 46]
```

```
426 {   9}   |   |     |     +-parseVerbatim()
            |   |     |       <void parseVerbatim () line:11849>
427 {  10}  |   |     |       +-getChar()
            |   |     |       | <int getChar () line:2490> [see 46]
428 {  10}  |   |     |       +-resizeVbuf()
429 {  10}  |   |     |       +-new_verb_node()
430 {  10}  |   |     |       +-fprintf()
431 {  10}  |   |     |       +-longjmp()
432 {  10}  |   |     |       +-strlen()
433 {  10}  |   |     |       +-allocString()
            |   |     |       | <char *allocString () line:2189> [see 61]
434 {  10}  |   |     |       \-allocNode()
            |   |     |         <TextNode *allocNode () line:9265> [see 419]
435 {   9}  |   |     |     +-parseFromString()
            |   |     |       <void parseFromString () line:9823> (R)
436 {  10}  |   |     |       +-saveScannerState()
            |   |     |        <void saveScannerState () line:2361>
437 {  11}  |   |     |         \-halloc() <char *halloc () line:2070>
            |   |     |           [see 8]
438 {  10}  |   |     |       +-parseHyperDoc()
            |   |     |       | <void parseHyperDoc () line:9938>
            |   |     |       | (recursive: see 421) [see 421]
439 {  10}  |   |     |       \-restoreScannerState()
            |   |     |         <void restoreScannerState () line:2377>
440 {  11}  |   |     |         +-fprintf()
441 {  11}  |   |     |         +-exit()
442 {  11}  |   |     |         +-fseek()
443 {  11}  |   |     |         \-free()
444 {   9}  |   |     |     \-makeLinkWindow()
            |   |     |       <HyperLink *makeLinkWindow () line:10639>
445 {  10}  |   |     |       +-printToString()
            |   |     |       | <char *printToString () line:13497> (R)
446 {  11}  |   |     |       | +-printToString1()
            |   |     |       |  <char *printToString1 () line:13504> (R)
447 {  12}  |   |     |       |   +-storeChar()
448 {  12}  |   |     |       |   +-checkCondition()
            |   |     |       |     <int checkCondition () line:3217> (R)
449 {  13}  |   |     |       |     +-hashFind()
            |   |     |       |     | <char *hashFind () line:2139> [see 68]
450 {  13}  |   |     |       |     +-strcmp()
451 {  13}  |   |     |       |     +-send_int()
452 {  13}  |   |     |       |     \-checkMemostack()
            |   |     |       |       <int checkMemostack () line:3199>
453 {  14}  |   |     |       |       +-printToString()
            |   |     |       |       | <char *printToString () line:13497>
            |   |     |       |       | (recursive: see 445) [see 445]
454 {  14}  |   |     |       |       \-strcmp()
455 {  12}  |   |     |       |   +-hashFind()
            |   |     |       |     <char *hashFind () line:2139> [see 68]
456 {  12}  |   |     |       |   +-fprintf()
```

```
457 {  12}   |   |     |       |   +-exit()
458 {  12}   |   |     |       |   +-returnItem()
             |   |     |       |   | <InputItem *returnItem () line:8613>
459 {  13}   |   |     |       |   | \-strcmp()
460 {  12}   |   |     |       |   +-funnyUnescape()
461 {  12}   |   |     |       |   +-atoi()
462 {  12}   |   |     |       |   \-strlen()
463 {  11}   |   |     |       | \-resizeBuffer()
             |   |     |       |   <char *resizeBuffer () line:9718>
464 {  12}   |   |     |       |   +-halloc()
             |   |     |       |   | <char *halloc () line:2070> [see 8]
465 {  12}   |   |     |       |   +-memset()
466 {  12}   |   |     |       |   +-memcpy()
467 {  12}   |   |     |       |   \-free()
468 {  10}   |   |     |       +-hashFind()
             |   |     |       | <char *hashFind () line:2139> [see 68]
469 {  10}   |   |     |       +-printf()
470 {  10}   |   |     |       +-halloc()
             |   |     |       | <char *halloc () line:2070> [see 8]
471 {  10}   |   |     |       +-fprintf()
472 {  10}   |   |     |       +-exit()
473 {  10}   |   |     |       +-XCreateWindow()
474 {  10}   |   |     |       \-hashInsert()
             |   |     |         <void hashInsert () line:2104> [see 24]
475 {   8}   |   |     |   +-parseHelp()
             |   |     |     <void parseHelp () line:12211> (R)
476 {   9}   |   |     |     +-getToken()
             |   |     |     | <int getToken () line:2535> (R) [see 74]
477 {   9}   |   |     |     +-tokenName()
             |   |     |     | <void tokenName () line:2204> [see 88]
478 {   9}   |   |     |     +-fprintf()
479 {   9}   |   |     |     +-printPageAndFilename()
             |   |     |     | <void printPageAndFilename () line:2286>
             |   |     |     | [see 92]
480 {   9}   |   |     |     +-jump() <void jump () line:2196> [see 108]
481 {   9}   |   |     |     +-free()
482 {   9}   |   |     |     +-allocString()
             |   |     |     | <char *allocString () line:2189> [see 61]
483 {   9}   |   |     |     \-getInputString()
             |   |     |       <char *getInputString () line:10837> (R)
484 {  10}   |   |     |       +-allocNode()
             |   |     |       | <TextNode *allocNode () line:9265> [see 419]
485 {  10}   |   |     |       +-parseHyperDoc()
             |   |     |       | <void parseHyperDoc () line:9938>
             |   |     |       | (recursive: see 421) [see 421]
486 {  10}   |   |     |       +-printToString()
             |   |     |       | <char *printToString () line:13497>
             |   |     |       | (R) [see 445]
487 {  10}   |   |     |       \-freeNode() <void freeNode () line:9281> (R)
488 {  11}   |   |     |         +-freePastearea()
```

```
                |   |     |         | <void freePastearea () line:9572>
489 {  12}      |   |     |         | +-hashFind()
                |   |     |         | | <char *hashFind () line:2139> [see 68]
490 {  12}      |   |     |         | +-hashDelete()
                |   |     |         | | <void hashDelete () line:2118>
491 {  12}      |   |     |         | +-freePaste()
                |   |     |         | | <void freePaste () line:9539>
492 {  13}      |   |     |         | | +-freeGroupStack()
                |   |     |         | | | <void freeGroupStack () line:7429>
493 {  14}      |   |     |         | | | \-free()
494 {  13}      |   |     |         | | +-freeItemStack()
                |   |     |         | | | <void freeItemStack () line:8703>
495 {  14}      |   |     |         | | | \-free()
496 {  13}      |   |     |         | | +-freeNode()
                |   |     |         | | | <void freeNode () line:9281>
                |   |     |         | | | (recursive: see 487) [see 487]
497 {  13}      |   |     |         | | \-free()
498 {  12}      |   |     |         | \-freeIfNonNULL()
                |   |     |         |   <void freeIfNonNULL () line:9201>
                |   |     |         |   [see 144]
499 {  11}      |   |     |         +-freeNode() <void freeNode () line:9281>
                |   |     |         | (recursive: see 487) [see 487]
500 {  11}      |   |     |         +-freePastebutton()
                |   |     |           <void freePastebutton () line:9548>
501 {  12}      |   |     |           +-hashFind()
                |   |     |           | <char *hashFind () line:2139> [see 68]
502 {  12}      |   |     |           +-hashDelete()
                |   |     |           | <void hashDelete () line:2118> [see 490]
503 {  12}      |   |     |           +-freePaste()
                |   |     |           | <void freePaste () line:9539> [see 491]
504 {  12}      |   |     |           +-XDestroyWindow()
505 {  12}      |   |     |           \-freeIfNonNULL()
                |   |     |             <void freeIfNonNULL () line:9201>
                |   |     |             [see 144]
506 {  11}      |   |     |         +-freeIfNonNULL()
                |   |     |         | <void freeIfNonNULL () line:9201>
                |   |     |         | [see 144]
507 {  11}      |   |     |         +-deleteItem() <int deleteItem () line:8624>
508 {  12}      |   |     |           +-strcmp()
509 {  12}      |   |     |           +-currentItem()
                |   |     |           | <InputItem *currentItem () line:11291>
510 {  12}      |   |     |           +-freeInputItem()
                |   |     |           | <void freeInputItem () line:9613>
                |   |     |           | [see 397]
511 {  12}      |   |     |           +-free()
512 {  12}      |   |     |           \-fprintf()
513 {  11}      |   |     |         +-hashDelete()
                |   |     |         | <void hashDelete () line:2118> [see 490]
514 {  11}      |   |     |         +-XDestroyWindow()
515 {  11}      |   |     |         \-free()
```

```
516 {   8}   |   |     |   +-parsePaste() <void parsePaste () line:11365> (R)
517 {   9}   |   |     |     +-fprintf()
518 {   9}   |   |     |     +-printPageAndFilename()
             |   |     |     | <void printPageAndFilename () line:2286>
             |   |     |     | [see 92]
519 {   9}   |   |     |     +-jump() <void jump () line:2196> [see 108]
520 {   9}   |   |     |     +-getToken()
             |   |     |     | <int getToken () line:2535> (R) [see 74]
521 {   9}   |   |     |     +-printNextTenTokens()
             |   |     |     | <void printNextTenTokens () line:2313>
             |   |     |     | (R) [see 95]
522 {   9}   |   |     |     +-allocString()
             |   |     |     | <char *allocString () line:2189> [see 61]
523 {   9}   |   |     |     +-getInputString()
             |   |     |     | <char *getInputString () line:10837>
             |   |     |     | (R) [see 483]
524 {   9}   |   |     |     +-hashFind()
             |   |     |     | <char *hashFind () line:2139> [see 68]
525 {   9}   |   |     |     +-allocPasteNode()
             |   |     |     | <PasteNode *allocPasteNode () line:9656>
526 {  10}   |   |     |     | +-halloc() <char *halloc () line:2070> [see 8]
527 {  10}   |   |     |     | \-allocString()
             |   |     |     |   <char *allocString () line:2189> [see 61]
528 {   9}   |   |     |     +-hashInsert()
             |   |     |     | <void hashInsert () line:2104> [see 24]
529 {   9}   |   |     |     +-currentItem()
             |   |     |     | <InputItem *currentItem () line:11291>
             |   |     |     | [see 509]
530 {   9}   |   |     |     +-getWhere() <int getWhere () line:10853>
531 {  10}   |   |     |     | +-getToken()
             |   |     |     | | <int getToken () line:2535> (R) [see 74]
532 {  10}   |   |     |     | \-strcmp()
533 {   9}   |   |     |     +-allocNode()
             |   |     |     | <TextNode *allocNode () line:9265> [see 419]
534 {   9}   |   |     |     \-parseHyperDoc()
             |   |     |       <void parseHyperDoc () line:9938>
             |   |     |       (recursive: see 421) [see 421]
535 {   8}   |   |     |   +-parsePastebutton()
             |   |     |     <void parsePastebutton () line:11445> (R)
536 {   9}   |   |     |     +-getToken()
             |   |     |     | <int getToken () line:2535> (R) [see 74]
537 {   9}   |   |     |     +-fprintf()
538 {   9}   |   |     |     +-printPageAndFilename()
             |   |     |     | <void printPageAndFilename () line:2286>
             |   |     |     | [see 92]
539 {   9}   |   |     |     +-printNextTenTokens()
             |   |     |     | <void printNextTenTokens () line:2313>
             |   |     |     | (R) [see 95]
540 {   9}   |   |     |     +-jump() <void jump () line:2196> [see 108]
541 {   9}   |   |     |     +-allocString()
```

```
             |   |     |     | <char *allocString () line:2189> [see 61]
542 {   9}   |   |     |     +-getInputString()
             |   |     |     | <char *getInputString () line:10837>
             |   |     |     | (R) [see 483]
543 {   9}   |   |     |     +-hashFind()
             |   |     |     | <char *hashFind () line:2139> [see 68]
544 {   9}   |   |     |     +-allocPasteNode()
             |   |     |     | <PasteNode *allocPasteNode () line:9656>
             |   |     |     | [see 525]
545 {   9}   |   |     |     +-hashInsert()
             |   |     |     | <void hashInsert () line:2104> [see 24]
546 {   9}   |   |     |     +-allocNode()
             |   |     |     | <TextNode *allocNode () line:9265> [see 419]
547 {   9}   |   |     |     +-parseHyperDoc()
             |   |     |     | <void parseHyperDoc () line:9938>
             |   |     |     | (recursive: see 421) [see 421]
548 {   9}   |   |     |     \-makePasteWindow()
             |   |     |       <HyperLink *makePasteWindow () line:10682>
549 {  10}   |   |     |       +-halloc() <char *halloc () line:2070> [see 8]
550 {  10}   |   |     |       +-fprintf()
551 {  10}   |   |     |       +-exit()
552 {  10}   |   |     |       +-XCreateWindow()
553 {  10}   |   |     |       \-hashInsert()
             |   |     |         <void hashInsert () line:2104> [see 24]
554 {   8}   |   |     |   +-endAPage() <void endAPage () line:10377>
555 {   9}   |   |     |     +-fprintf()
556 {   9}   |   |     |     +-printPageAndFilename()
             |   |     |     | <void printPageAndFilename () line:2286>
             |   |     |     | [see 92]
557 {   9}   |   |     |     +-jump() <void jump () line:2196> [see 108]
558 {   9}   |   |     |     \-PopMR() <void PopMR () line:9743>
559 {  10}   |   |     |       +-fprintf()
560 {  10}   |   |     |       +-exit()
561 {  10}   |   |     |       \-free()
562 {   8}   |   |     |   +-startFooter() <void startFooter () line:10352>
563 {   9}   |   |     |     +-fprintf()
564 {   9}   |   |     |     +-printPageAndFilename()
             |   |     |     | <void printPageAndFilename () line:2286>
             |   |     |     | [see 92]
565 {   9}   |   |     |     +-longjmp()
566 {   9}   |   |     |     +-PopMR() <void PopMR () line:9743> [see 558]
567 {   9}   |   |     |     +-linkScrollBars()
             |   |     |     | <void linkScrollBars () line:12640>
568 {  10}   |   |     |     | +-halloc() <char *halloc () line:2070> [see 8]
569 {  10}   |   |     |     | \-hashInsert()
             |   |     |     |   <void hashInsert () line:2104> [see 24]
570 {   9}   |   |     |     +-PushMR() <void PushMR () line:9734> [see 416]
571 {   9}   |   |     |     \-allocNode()
             |   |     |       <TextNode *allocNode () line:9265> [see 419]
572 {   8}   |   |     |   +-startScrolling()
```

```
                |   |     |     <void startScrolling () line:10329>
573 {   9}      |   |     |     +-fprintf()
574 {   9}      |   |     |     +-longjmp()
575 {   9}      |   |     |     +-PopMR() <void PopMR () line:9743> [see 558]
576 {   9}      |   |     |     +-PushMR() <void PushMR () line:9734> [see 416]
577 {   9}      |   |     |     \-allocNode()
                |   |     |       <TextNode *allocNode () line:9265> [see 419]
578 {   8}      |   |     |   +-allocString()
                |   |     |     <char *allocString () line:2189> [see 61]
579 {   8}      |   |     |   +-parseNewcond() <void parseNewcond () line:11755>
580 {   9}      |   |     |     +-getExpectedToken()
                |   |     |      <void getExpectedToken () line:2406>
                |   |     |      (R) [see 86]
581 {   9}      |   |     |     +-strcpy()
582 {   9}      |   |     |     \-insertCond() <void insertCond () line:3172>
583 {  10}      |   |     |       +-hashFind()
                |   |     |       | <char *hashFind () line:2139> [see 68]
584 {  10}      |   |     |       +-fprintf()
585 {  10}      |   |     |       +-printPageAndFilename()
                |   |     |       | <void printPageAndFilename () line:2286>
                |   |     |       | [see 92]
586 {  10}      |   |     |       +-jump() <void jump () line:2196> [see 108]
587 {  10}      |   |     |       +-allocCondnode()
                |   |     |       | <CondNode *allocCondnode () line:9456>
588 {  11}      |   |     |       | \-halloc()
                |   |     |       |   <char *halloc () line:2070> [see 8]
589 {  10}      |   |     |       +-halloc()
                |   |     |       | <char *halloc () line:2070> [see 8]
590 {  10}      |   |     |       +-strlen()
591 {  10}      |   |     |       +-strcpy()
592 {  10}      |   |     |       \-hashInsert()
                |   |     |         <void hashInsert () line:2104> [see 24]
593 {   8}      |   |     |   +-parseSetcond() <void parseSetcond () line:11765>
594 {   9}      |   |     |     +-getExpectedToken()
                |   |     |       <void getExpectedToken () line:2406>
                |   |     |       (R) [see 86]
595 {   9}      |   |     |     +-strcpy()
596 {   9}      |   |     |     \-changeCond() <void changeCond () line:3187>
597 {  10}      |   |     |       +-hashFind()
                |   |     |       | <char *hashFind () line:2139> [see 68]
598 {  10}      |   |     |       +-fprintf()
599 {  10}      |   |     |       +-free()
600 {  10}      |   |     |       +-halloc() <char *halloc () line:2070> [see 8]
601 {  10}      |   |     |       +-strlen()
602 {  10}      |   |     |       \-strcpy()
603 {   8}      |   |     |   +-parseVerbatim()
                |   |     |     <void parseVerbatim () line:11849> [see 426]
604 {   8}      |   |     |   +-parseIfcond()
                |   |     |     <void parseIfcond () line:11647> (R)
605 {   9}      |   |     |     +-fprintf()
```

```
606 {   9}   |   |     |     +-longjmp()
607 {   9}   |   |     |     +-exit()
608 {   9}   |   |     |     +-allocIfnode()
             |   |     |     | <IfNode *allocIfnode () line:9449>
609 {  10}   |   |     |     | \-halloc() <char *halloc () line:2070> [see 8]
610 {   9}   |   |     |     +-allocNode()
             |   |     |     | <TextNode *allocNode () line:9265> [see 419]
611 {   9}   |   |     |     +-parseCondnode()
             |   |     |       <void parseCondnode () line:11713>
612 {  10}   |   |     |       +-getToken()
             |   |     |       | <int getToken () line:2535> (R) [see 74]
613 {  10}   |   |     |       +-allocString()
             |   |     |       | <char *allocString () line:2189> [see 61]
614 {  10}   |   |     |       +-parseHasreturnto()
             |   |     |       | <void parseHasreturnto () line:11744>
615 {  11}   |   |     |       | +-allocNode()
             |   |     |       | | <TextNode *allocNode () line:9265>
             |   |     |       | | [see 419]
616 {  11}   |   |     |       | +-getExpectedToken()
             |   |     |       | | <void getExpectedToken () line:2406>
             |   |     |       | | (R) [see 86]
617 {  11}   |   |     |       | \-parseHyperDoc()
             |   |     |       |   <void parseHyperDoc () line:9938>
             |   |     |       |   (recursive: see 421) [see 421]
618 {  10}   |   |     |       +-tokenName()
             |   |     |       | <void tokenName () line:2204> [see 88]
619 {  10}   |   |     |       +-sprintf()
620 {  10}   |   |     |       \-tpderror() <void tpderror () line:2979>
621 {  11}   |   |     |         +-sprintf()
622 {  11}   |   |     |         +-fprintf()
623 {  11}   |   |     |         +-printPageAndFilename()
             |   |     |         | <void printPageAndFilename () line:2286>
             |   |     |         | [see 92]
624 {  11}   |   |     |         \-printNextTenTokens()
             |   |     |           <void printNextTenTokens () line:2313>
             |   |     |           (R) [see 95]
625 {   9}   |   |     |     +-parseHyperDoc()
             |   |     |     | <void parseHyperDoc () line:9938>
             |   |     |     | (recursive: see 421) [see 421]
626 {   9}   |   |     |     \-tokenName()
             |   |     |       <void tokenName () line:2204> [see 88]
627 {   8}   |   |     |   +-fprintf()
628 {   8}   |   |     |   +-longjmp()
629 {   8}   |   |     |   +-exit()
630 {   8}   |   |     |   +-parseMacro() <int parseMacro () line:9062> (R)
631 {   9}   |   |     |     +-allocNode()
             |   |     |     | <TextNode *allocNode () line:9265> [see 419]
632 {   9}   |   |     |     +-hashFind()
             |   |     |     | <char *hashFind () line:2139> [see 68]
633 {   9}   |   |     |     +-loadMacro() <char *loadMacro () line:8954>
```

```
634 {  10}   |   |     |     | +-saveScannerState()
             |   |     |     | | <void saveScannerState () line:2361>
             |   |     |     | | [see 436]
635 {  10}   |   |     |     | +-findFp()
             |   |     |     | | <FILE *findFp () line:10878> [see 365]
636 {  10}   |   |     |     | +-initScanner()
             |   |     |     | | <void initScanner () line:2343> [see 40]
637 {  10}   |   |     |     | +-getExpectedToken()
             |   |     |     | | <void getExpectedToken () line:2406>
             |   |     |     | | (R) [see 86]
638 {  10}   |   |     |     | +-strcmp()
639 {  10}   |   |     |     | +-fprintf()
640 {  10}   |   |     |     | +-longjmp()
641 {  10}   |   |     |     | +-getToken()
             |   |     |     | | <int getToken () line:2535> (R) [see 74]
642 {  10}   |   |     |     | +-number() <int number () line:8946>
643 {  11}   |   |     |     | | \-isdigit()
644 {  10}   |   |     |     | +-atoi()
645 {  10}   |   |     |     | +-scanHyperDoc()
             |   |     |     | | <void scanHyperDoc () line:8914>
646 {  11}   |   |     |     | | +-getToken()
             |   |     |     | | |   <int getToken () line:2535> (R) [see 74]
647 {  11}   |   |     |     | | +-fprintf()
648 {  11}   |   |     |     | | \-longjmp()
649 {  10}   |   |     |     | +-fseek()
650 {  10}   |   |     |     | +-halloc() <char *halloc () line:2070> [see 8]
651 {  10}   |   |     |     | +-getc()
652 {  10}   |   |     |     | \-restoreScannerState()
             |   |     |     |   <void restoreScannerState () line:2377>
             |   |     |     |   [see 439]
653 {   9}   |   |     |     +-getParameterStrings()
             |   |     |     | <void getParameterStrings () line:9098>
654 {  10}   |   |     |     | +-initParameterElem()
             |   |     |     | <ParameterList initParameterElem () line:9017>
655 {  11}   |   |     |     | | \-halloc() <char *halloc () line:2070>
             |   |     |     | |  [see 8]
656 {  10}   |   |     |     | +-pushParameters()
             |   |     |     | | <int pushParameters () line:9034>
657 {  11}   |   |     |     | | +-fprintf()
658 {  11}   |   |     |     | | \-longjmp()
659 {  10}   |   |     |     | +-getToken()
             |   |     |     | | <int getToken () line:2535> (R) [see 74]
660 {  10}   |   |     |     | +-fprintf()
661 {  10}   |   |     |     | +-jump() <void jump () line:2196> [see 108]
662 {  10}   |   |     |     | +-getChar()
             |   |     |       | <int getChar () line:2490> [see 46]
663 {  10}   |   |     |     | +-longjmp()
664 {  10}   |   |     |     | +-numeric()
665 {  10}   |   |     |     | +-ungetChar()
             |   |     |       | <void ungetChar () line:2400> [see 60]
```

```
666 {  10}   |   |     |     | +-atoi()
667 {  10}   |   |     |     | +-strlen()
668 {  10}   |   |     |     | +-halloc() <char *halloc () line:2070> [see 8]
669 {  10}   |   |     |     | \-strcpy()
670 {   9}   |   |     |     +-parseFromString()
             |   |     |     | <void parseFromString () line:9823>
             |   |     |     | (R) [see 435]
671 {   9}   |   |     |     +-popParameters()
             |   |     |     | <int popParameters () line:9044>
672 {  10}   |   |     |     | \-free()
673 {   9}   |   |     |     +-fprintf()
674 {   9}   |   |     |     \-longjmp()
675 {   8}   |   |     |   +-parseEnv() <void parseEnv () line:12065>
676 {   9}   |   |     |     +-getExpectedToken()
             |   |     |     | <void getExpectedToken () line:2406>
             |   |     |     | (R) [see 86]
677 {   9}   |   |     |     +-getenv()
678 {   9}   |   |     |     +-fprintf()
679 {   9}   |   |     |     +-halloc() <char *halloc () line:2070> [see 8]
680 {   9}   |   |     |     +-strcpy()
681 {   9}   |   |     |     +-free()
682 {   9}   |   |     |     \-allocString()
             |   |     |       <char *allocString () line:2189> [see 61]
683 {   8}   |   |     |   +-windowId() <char *windowId () line:10417>
684 {   9}   |   |     |     +-sprintf()
685 {   9}   |   |     |     +-strlen()
686 {   9}   |   |     |     +-halloc() <char *halloc () line:2070> [see 8]
687 {   9}   |   |     |     \-strcpy()
688 {   8}   |   |     |   +-halloc() <char *halloc () line:2070> [see 8]
689 {   8}   |   |     |   +-strlen()
690 {   8}   |   |     |   +-sprintf()
691 {   8}   |   |     |   +-parseBeginItems()
             |   |     |     <void parseBeginItems () line:11779> (R)
692 {   9}   |   |     |     +-getToken()
             |   |     |     | <int getToken () line:2535> (R) [see 74]
693 {   9}   |   |     |     +-allocNode()
             |   |     |     | <TextNode *allocNode () line:9265> [see 419]
694 {   9}   |   |     |     +-parseHyperDoc()
             |   |     |     | <void parseHyperDoc () line:9938>
             |   |     |     | (recursive: see 421) [see 421]
695 {   9}   |   |     |     +-fprintf()
696 {   9}   |   |     |     +-printNextTenTokens()
             |   |     |     | <void printNextTenTokens () line:2313>
             |   |     |     | (R) [see 95]
697 {   9}   |   |     |     +-printPageAndFilename()
             |   |     |     | <void printPageAndFilename () line:2286>
             |   |     |     | [see 92]
698 {   9}   |   |     |     +-jump() <void jump () line:2196> [see 108]
699 {   9}   |   |     |     \-ungetToken() <void ungetToken () line:2427>
700 {  10}   |   |     |       \-allocString()
```

```
                |   |     |         <char *allocString () line:2189> [see 61]
701 {   8}      |   |     |   +-parseItem() <void parseItem () line:11807> (R)
702 {   9}      |   |     |     +-fprintf()
703 {   9}      |   |     |     +-printPageAndFilename()
                |   |     |     | <void printPageAndFilename () line:2286>
                |   |     |     | [see 92]
704 {   9}      |   |     |     +-printNextTenTokens()
                |   |     |     | <void printNextTenTokens () line:2313>
                |   |     |     | (R) [see 95]
705 {   9}      |   |     |     +-jump() <void jump () line:2196> [see 108]
706 {   9}      |   |     |     +-getToken()
                |   |     |     | <int getToken () line:2535> (R) [see 74]
707 {   9}      |   |     |     +-allocNode()
                |   |     |     | <TextNode *allocNode () line:9265> [see 419]
708 {   9}      |   |     |     +-parseHyperDoc()
                |   |     |     | <void parseHyperDoc () line:9938>
                |   |     |     | (recursive: see 421) [see 421]
709 {   9}      |   |     |     \-ungetToken()
                |   |     |       <void ungetToken () line:2427> [see 699]
710 {   8}      |   |     |   +-parseMitem() <void parseMitem () line:11839>
711 {   9}      |   |     |     +-fprintf()
712 {   9}      |   |     |     +-printPageAndFilename()
                |   |     |     | <void printPageAndFilename () line:2286>
                |   |     |     | [see 92]
713 {   9}      |   |     |     +-printNextTenTokens()
                |   |     |     | <void printNextTenTokens () line:2313>
                |   |     |     | (R) [see 95]
714 {   9}      |   |     |     \-jump() <void jump () line:2196> [see 108]
715 {   8}      |   |     |   +-parseValue1()
                |   |     |     <void parseValue1 () line:12091> (R)
716 {   9}      |   |     |     +-allocNode()
                |   |     |     | <TextNode *allocNode () line:9265> [see 419]
717 {   9}      |   |     |     +-getExpectedToken()
                |   |     |     | <void getExpectedToken () line:2406>
                |   |     |     | (R) [see 86]
718 {   9}      |   |     |     +-getInputString()
                |   |     |     | <char *getInputString () line:10837>
                |   |     |     | (R) [see 483]
719 {   9}      |   |     |     +-isNumber() <int isNumber () line:10772>
720 {  10}      |   |     |     | \-isdigit()
721 {   9}      |   |     |     +-fprintf()
722 {   9}      |   |     |     +-strcpy()
723 {   9}      |   |     |     \-allocString()
                |   |     |       <char *allocString () line:2189> [see 61]
724 {   8}      |   |     |   +-parseValue2()
                |   |     |     <void parseValue2 () line:12112> (R)
725 {   9}      |   |     |     +-allocNode()
                |   |     |     | <TextNode *allocNode () line:9265> [see 419]
726 {   9}      |   |     |     +-getExpectedToken()
                |   |     |     | <void getExpectedToken () line:2406>
```

```
             |   |     |     | (R) [see 86]
727 {   9}   |   |     |     +-getInputString()
             |   |     |     | <char *getInputString () line:10837>
             |   |     |     | (R) [see 483]
728 {   9}   |   |     |     +-isNumber()
             |   |     |     | <int isNumber () line:10772> [see 719]
729 {   9}   |   |     |     +-fprintf()
730 {   9}   |   |     |     +-strcpy()
731 {   9}   |   |     |     \-allocString()
             |   |     |       <char *allocString () line:2189> [see 61]
732 {   8}   |   |     |   +-pushGroupStack()
             |   |     |   | <void pushGroupStack () line:7312>
733 {   9}   |   |     |   | \-halloc() <char *halloc () line:2070> [see 8]
734 {   8}   |   |     |   +-allocNode()
             |   |     |   | <TextNode *allocNode () line:9265> [see 419]
735 {   8}   |   |     |   +-parseHyperDoc()
             |   |     |   | <void parseHyperDoc () line:9938>
             |   |     |   | (recursive: see 421) [see 421]
736 {   8}   |   |     |   +-popGroupStack()
             |   |     |   | <int popGroupStack () line:7294> [see 404]
737 {   8}   |   |     |   +-parseButton()
             |   |     |   | <void parseButton () line:11982> (R)
738 {   9}   |   |     |   | +-fprintf()
739 {   9}   |   |     |   | +-longjmp()
740 {   9}   |   |     |   | +-allocNode()
             |   |     |   | | <TextNode *allocNode () line:9265> [see 419]
741 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | | <void getExpectedToken () line:2406>
             |   |     |   | | (R) [see 86]
742 {   9}   |   |     |   | +-parseHyperDoc()
             |   |     |   | | <void parseHyperDoc () line:9938>
             |   |     |   | | (recursive: see 421) [see 421]
743 {   9}   |   |     |   | \-makeLinkWindow()
             |   |     |   |   <HyperLink *makeLinkWindow () line:10639>
             |   |     |   |   [see 444]
744 {   8}   |   |     |   +-parseCommand()
             |   |     |   | <void parseCommand () line:11950> (R)
745 {   9}   |   |     |   | +-fprintf()
746 {   9}   |   |     |   | +-longjmp()
747 {   9}   |   |     |   | +-allocNode()
             |   |     |   | | <TextNode *allocNode () line:9265> [see 419]
748 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | | <void getExpectedToken () line:2406>
             |   |     |   | | (R) [see 86]
749 {   9}   |   |     |   | +-parseHyperDoc()
             |   |     |   | | <void parseHyperDoc () line:9938>
             |   |     |   | | (recursive: see 421) [see 421]
750 {   9}   |   |     |   | \-makeLinkWindow()
             |   |     |   |   <HyperLink *makeLinkWindow () line:10639>
             |   |     |   |   [see 444]
```

```
751 {   8}   |   |     |   +-parseInputPix()
             |   |     |   | <void parseInputPix () line:11898> (R)
752 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | | <void getExpectedToken () line:2406>
             |   |     |   | | (R) [see 86]
753 {   9}   |   |     |   | +-getInputString()
             |   |     |   | | <char *getInputString () line:10837>
             |   |     |   | | (R) [see 483]
754 {   9}   |   |     |   | +-allocString()
             |   |     |   | | <char *allocString () line:2189> [see 61]
755 {   9}   |   |     |   | +-DisplayPlanes()
756 {   9}   |   |     |   | +-strcpy()
757 {   9}   |   |     |   | +-strcat()
758 {   9}   |   |     |   | \-free()
759 {   8}   |   |     |   +-parseBox() <void parseBox () line:12176> (R)
760 {   9}   |   |     |   | +-allocNode()
             |   |     |   | | <TextNode *allocNode () line:9265> [see 419]
761 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | | <void getExpectedToken () line:2406>
             |   |     |   | | (R) [see 86]
762 {   9}   |   |     |   | \-parseHyperDoc()
             |   |     |   |   <void parseHyperDoc () line:9938>
             |   |     |   |   (recursive: see 421) [see 421]
763 {   8}   |   |     |   +-parseMbox() <void parseMbox () line:12187> (R)
764 {   9}   |   |     |   | +-allocNode()
             |   |     |   | | <TextNode *allocNode () line:9265>
             |   |     |   | | [see 419]
765 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | |<void getExpectedToken () line:2406>
             |   |     |   | |(R) [see 86]
766 {   9}   |   |     |   | \-parseHyperDoc()
             |   |     |   |   <void parseHyperDoc () line:9938>
             |   |     |   |   (recursive: see 421) [see 421]
767 {   8}   |   |     |   +-parseFree() <void parseFree () line:12198> (R)
768 {   9}   |   |     |   | +-allocNode()
             |   |     |   | | <TextNode *allocNode () line:9265> [see 419]
769 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | | <void getExpectedToken () line:2406>
             |   |     |   | | (R) [see 86]
770 {   9}   |   |     |   | \-parseHyperDoc()
             |   |     |   |   <void parseHyperDoc () line:9938>
             |   |     |   |   (recursive: see 421) [see 421]
771 {   8}   |   |     |   +-parseCenterline()
             |   |     |   | <void parseCenterline () line:11932> (R)
772 {   9}   |   |     |   | +-allocNode()
             |   |     |   | | <TextNode *allocNode () line:9265> [see 419]
773 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | | <void getExpectedToken () line:2406>
             |   |     |   | | (R) [see 86]
774 {   9}   |   |     |   | +-parseHyperDoc()
```

```
              |   |     |   | | <void parseHyperDoc () line:9938>
              |   |     |   | | (recursive: see 421) [see 421]
775 {   9}    |   |     |   | +-fprintf()
776 {   9}    |   |     |   | +-printPageAndFilename()
              |   |     |   | | <void printPageAndFilename () line:2286>
              |   |     |   | | [see 92]
777 {   9}    |   |     |   | +-printNextTenTokens()
              |   |     |   | | <void printNextTenTokens () line:2313>
              |   |     |   | | (R) [see 95]
778 {   9}    |   |     |   | \-longjmp()
779 {   8}    |   |     |   +-addDependencies()
              |   |     |   | <void addDependencies () line:10732> (R)
780 {   9}    |   |     |   | +-fprintf()
781 {   9}    |   |     |   | +-printPageAndFilename()
              |   |     |   | | <void printPageAndFilename () line:2286>
              |   |     |   | | [see 92]
782 {   9}    |   |     |   | +-exit()
783 {   9}    |   |     |   | +-allocNode()
              |   |     |   | | <TextNode *allocNode () line:9265> [see 419]
784 {   9}    |   |     |   | +-getExpectedToken()
              |   |     |   | | <void getExpectedToken () line:2406>
              |   |     |   | | (R) [see 86]
785 {   9}    |   |     |   | +-parseHyperDoc()
              |   |     |   | | <void parseHyperDoc () line:9938>
              |   |     |   | | (recursive: see 421) [see 421]
786 {   9}    |   |     |   | +-halloc() <char *halloc () line:2070> [see 8]
787 {   9}    |   |     |   | +-hashInit()
              |   |     |   | | <void hashInit () line:2091> [see 7]
788 {   9}    |   |     |   | +-stringEqual()
              |   |     |   | | <int stringEqual () line:2185> [see 14]
789 {   9}    |   |     |   | +-stringHash() <int stringHash () line:2177>
790 {   9}    |   |     |   | +-allocString()
              |   |     |   | | <char *allocString () line:2189> [see 61]
791 {   9}    |   |     |   | \-hashInsert()
              |   |     |   |   <void hashInsert () line:2104> [see 24]
792 {   8}    |   |     |   +-parseSpadcommand()
              |   |     |   |  <void parseSpadcommand () line:12018> (R)
793 {   9}    |   |     |   | +-getExpectedToken()
              |   |     |   | | <void getExpectedToken () line:2406>
              |   |     |   | | (R) [see 86]
794 {   9}    |   |     |   | +-allocNode()
              |   |     |   | | <TextNode *allocNode () line:9265> [see 419]
795 {   9}    |   |     |   | +-parseHyperDoc()
              |   |     |   | | <void parseHyperDoc () line:9938>
              |   |     |   | | (recursive: see 421) [see 421]
796 {   9}    |   |     |   | \-makeLinkWindow()
              |   |     |   |   <HyperLink *makeLinkWindow () line:10639>
              |   |     |   |   [see 444]
797 {   8}    |   |     |   +-parseTable() <void parseTable () line:12134> (R)
798 {   9}    |   |     |   | +-fprintf()
```

```
799 {   9}   |   |     |   | +-longjmp()
800 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | | <void getExpectedToken () line:2406>
             |   |     |   | | (R) [see 86]
801 {   9}   |   |     |   | +-allocNode()
             |   |     |   | | <TextNode *allocNode () line:9265> [see 419]
802 {   9}   |   |     |   | +-getToken()
             |   |     |   | | <int getToken () line:2535> (R) [see 74]
803 {   9}   |   |     |   | +-parseHyperDoc()
             |   |     |   | | <void parseHyperDoc () line:9938>
             |   |     |   | | (recursive: see 421) [see 421]
804 {   9}   |   |     |   | +-tokenName()
             |   |     |   | | <void tokenName () line:2204> [see 88]
805 {   9}   |   |     |   | +-printPageAndFilename()
             |   |     |   | | <void printPageAndFilename () line:2286>
             |   |     |   | | [see 92]
806 {   9}   |   |     |   | +-jump() <void jump () line:2196> [see 108]
807 {   9}   |   |     |   | \-free()
808 {   8}   |   |     |   +-parseInputstring()
             |   |     |   | <void parseInputstring () line:10996> (R)
809 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | | <void getExpectedToken () line:2406>
             |   |     |   | | (R) [see 86]
810 {   9}   |   |     |   | +-getInputString()
             |   |     |   | | <char *getInputString () line:10837>
             |   |     |   | | (R) [see 483]
811 {   9}   |   |     |   | +-allocString()
             |   |     |   | | <char *allocString () line:2189> [see 61]
812 {   9}   |   |     |   | +-atoi()
813 {   9}   |   |     |   | +-fprintf()
814 {   9}   |   |     |   | +-longjmp()
815 {   9}   |   |     |   | +-halloc() <char *halloc () line:2070> [see 8]
816 {   9}   |   |     |   | +-strlen()
817 {   9}   |   |     |   | +-strcpy()
818 {   9}   |   |     |   | +-initializeDefault()
             |   |     |   | | <void initializeDefault () line:10955>
819 {  10}   |   |     |   | | \-allocInputline()
             |   |     |   | |   <LineStruct *allocInputline () line:9644>
820 {  11}   |   |     |   | |   \-halloc()
             |   |     |   | |     <char *halloc () line:2070> [see 8]
821 {   9}   |   |     |   | +-makeInputWindow()
             |   |     |   |   <HyperLink *makeInputWindow () line:10900>
822 {  10}   |   |     |   |   +-halloc() <char *halloc () line:2070> [see 8]
823 {  10}   |   |     |   |   +-fprintf()
824 {  10}   |   |     |   |   +-exit()
825 {  10}   |   |     |   |   +-XCreateWindow()
826 {  10}   |   |     |   |   +-XSelectInput()
827 {  10}   |   |     |   |   \-hashInsert()
             |   |     |   |     <void hashInsert () line:2104> [see 24]
828 {   9}   |   |     |   | \-insertItem() <void insertItem () line:11243>
```

```
829 {   8}   |   |     |   +-parseSimplebox()
             |   |     |   | <void parseSimplebox () line:11042> (R)
830 {   9}   |   |     |   | +-getToken()
             |   |     |   | | <int getToken () line:2535> (R) [see 74]
831 {   9}   |   |     |   | +-getExpectedToken()
             |   |     |   | | <void getExpectedToken () line:2406>
             |   |     |   | | (R) [see 86]
832 {   9}   |   |     |   | +-isNumber()
             |   |     |   | | <int isNumber () line:10772> [see 719]
833 {   9}   |   |     |   | +-fprintf()
834 {   9}   |   |     |   | +-printPageAndFilename()
             |   |     |   | | <void printPageAndFilename () line:2286>
             |   |     |   | | [see 92]
835 {   9}   |   |     |   | +-jump() <void jump () line:2196> [see 108]
836 {   9}   |   |     |   | +-strcmp()
837 {   9}   |   |     |   | +-tokenName()
             |   |     |   | | <void tokenName () line:2204> [see 88]
838 {   9}   |   |     |   | +-getInputString()
             |   |     |   | | <char *getInputString () line:10837>
             |   |     |   | | (R) [see 483]
839 {   9}   |   |     |   | +-hashFind()
             |   |     |   | | <char *hashFind () line:2139> [see 68]
840 {   9}   |   |     |   | +-allocInputbox()
             |   |     |   | | <InputBox *allocInputbox () line:9687>
841 {  10}   |   |     |   | | \-halloc() <char *halloc () line:2070> [see 8]
842 {   9}   |   |     |   | +-allocString()
             |   |     |   | | <char *allocString () line:2189> [see 61]
843 {   9}   |   |     |   | +-insertImageStruct()
             |   |     |   | | <ImageStruct *insertImageStruct () line:12365>
844 {  10}   |   |     |   | | +-hashFind()
             |   |     |   | | | <char *hashFind () line:2139> [see 68]
845 {  10}   |   |     |   | | +-HTReadBitmapFile()
             |   |     |   | | | <XImage *HTReadBitmapFile () line:12233>
             |   |     |   | | | [see 266]
846 {  10}   |   |     |   | | +-halloc() <char *halloc () line:2070> [see 8]
847 {  10}   |   |     |   | | +-strlen()
848 {  10}   |   |     |   | | +-sprintf()
849 {  10}   |   |     |   | | \-hashInsert()
             |   |     |   | |   <void hashInsert () line:2104> [see 24]
850 {   9}   |   |     |   | +-max()
851 {   9}   |   |     |   | +-makeBoxWindow()
             |   |     |   |   <HyperLink *makeBoxWindow () line:10929>
852 {  10}   |   |     |   |   +-halloc() <char *halloc () line:2070> [see 8]
853 {  10}   |   |     |   |   +-fprintf()
854 {  10}   |   |     |   |   +-exit()
855 {  10}   |   |     |   |   +-XCreateWindow()
856 {  10}   |   |     |   |   +-XSelectInput()
857 {  10}   |   |     |   |   \-hashInsert()
             |   |     |   |     <void hashInsert () line:2104> [see 24]
858 {   9}   |   |     |   | +-halloc() <char *halloc () line:2070> [see 8]
```

```
859 {   9}   |   |     |   | +-hashInit()
             |   |     |   | | <void hashInit () line:2091> [see 7]
860 {   9}   |   |     |   | +-stringEqual()
             |   |     |   | | <int stringEqual () line:2185> [see 14]
861 {   9}   |   |     |   | +-stringHash() <int stringHash () line:2177>
862 {   9}   |   |     |   | \-hashInsert()
             |   |     |   |   <void hashInsert () line:2104> [see 24]
863 {   8}   |   |     |   +-strcpy()
864 {   8}   |   |     |   +-strcat()
865 {   8}   |   |     |   +-parserError() <void parserError () line:10781>
866 {   9}   |   |     |   | +-fprintf()
867 {   9}   |   |     |   | +-getToken()
             |   |     |   |   <int getToken () line:2535> (R) [see 74]
868 {   9}   |   |     |   | +-printToken()
             |   |     |   |   <void printToken () line:2276> [see 98]
869 {   9}   |   |     |   | \-exit()
870 {   8}   |   |     |   +-getExpectedToken()
             |   |     |     <void getExpectedToken () line:2406>
             |   |     |     (R) [see 86]
871 {   8}   |   |     |   +-parseParameters()
             |   |     |     <void parseParameters () line:9182> (R)
872 {   9}   |   |     |     +-number() <int number () line:8946> [see 642]
873 {   9}   |   |     |     +-fprintf()
874 {   9}   |   |     |     +-longjmp()
875 {   9}   |   |     |     +-atoi()
876 {   9}   |   |     |     \-parseFromString()
             |   |     |       <void parseFromString () line:9823>
             |   |     |       (R) [see 435]
877 {   8}   |   |     |   +-parseRadiobox()
             |   |     |     <void parseRadiobox () line:11122> (R)
878 {   9}   |   |     |     +-getToken()
             |   |     |     | <int getToken () line:2535> (R) [see 74]
879 {   9}   |   |     |     +-getExpectedToken()
             |   |     |     | <void getExpectedToken () line:2406>
             |   |     |     | (R) [see 86]
880 {   9}   |   |     |     +-isNumber()
             |   |     |     | <int isNumber () line:10772> [see 719]
881 {   9}   |   |     |     +-fprintf()
882 {   9}   |   |     |     +-printPageAndFilename()
             |   |     |     | <void printPageAndFilename () line:2286>
             |   |     |     | [see 92]
883 {   9}   |   |     |     +-jump() <void jump () line:2196> [see 108]
884 {   9}   |   |     |     +-strcmp()
885 {   9}   |   |     |     +-tokenName()
             |   |     |     | <void tokenName () line:2204> [see 88]
886 {   9}   |   |     |     +-getInputString()
             |   |     |     | <char *getInputString () line:10837>
             |   |     |     | (R) [see 483]
887 {   9}   |   |     |     +-hashFind()
             |   |     |     | <char *hashFind () line:2139> [see 68]
```

```
888 {   9}   |   |     |     +-allocInputbox()
             |   |     |     | <InputBox *allocInputbox () line:9687>
             |   |     |     | [see 840]
889 {   9}   |   |     |     +-allocString()
             |   |     |     | <char *allocString () line:2189>
             |   |     |     | [see 61]
890 {   9}   |   |     |     +-addBoxToRbList()
             |   |     |     | <void addBoxToRbList () line:11206>
891 {  10}   |   |     |     | +-strcmp()
892 {  10}   |   |     |     | +-fprintf()
893 {  10}   |   |     |     | +-printPageAndFilename()
             |   |     |     | | <void printPageAndFilename () line:2286>
             |   |     |     | | [see 92]
894 {  10}   |   |     |     | +-jump() <void jump () line:2196> [see 108]
895 {  10}   |   |     |     | \-checkOthers()
             |   |     |     |   <int checkOthers () line:11233>
896 {   9}   |   |     |     +-makeBoxWindow()
             |   |     |     | <HyperLink *makeBoxWindow () line:10929>
             |   |     |     | [see 851]
897 {   9}   |   |     |     +-halloc() <char *halloc () line:2070> [see 8]
898 {   9}   |   |     |     +-hashInit()
             |   |     |     | <void hashInit () line:2091> [see 7]
899 {   9}   |   |     |     +-stringEqual()
             |   |     |     | <int stringEqual () line:2185> [see 14]
900 {   9}   |   |     |     +-stringHash() <int stringHash () line:2177>
901 {   9}   |   |     |     \-hashInsert()
             |   |     |       <void hashInsert () line:2104> [see 24]
902 {   8}   |   |     |   +-parseRadioboxes()
             |   |     |     <void parseRadioboxes () line:11311> (R)
903 {   9}   |   |     |     +-allocRbs() <RadioBoxes *allocRbs () line:9695>
904 {  10}   |   |     |     | \-halloc() <char *halloc () line:2070> [see 8]
905 {   9}   |   |     |     +-getToken()
             |   |     |     | <int getToken () line:2535> (R) [see 74]
906 {   9}   |   |     |     +-tokenName()
             |   |     |     | <void tokenName () line:2204> [see 88]
907 {   9}   |   |     |     +-fprintf()
908 {   9}   |   |     |     +-printPageAndFilename()
             |   |     |     | <void printPageAndFilename () line:2286>
             |   |     |     | [see 92]
909 {   9}   |   |     |     +-jump() <void jump () line:2196> [see 108]
910 {   9}   |   |     |     +-allocString()
             |   |     |     | <char *allocString () line:2189> [see 61]
911 {   9}   |   |     |     +-getInputString()
             |   |     |     | <char *getInputString () line:10837>
             |   |     |     | (R) [see 483]
912 {   9}   |   |     |     +-alreadyThere()
             |   |     |     | <int alreadyThere () line:11301>
913 {  10}   |   |     |     | \-strcmp()
914 {   9}   |   |     |     +-free()
915 {   9}   |   |     |     +-insertImageStruct()
```

```
              |   |     |     |<ImageStruct *insertImageStruct () line:12365>
              |   |     |     |[see 843]
916 {   9}    |   |     |     \-max()
917 {   8}    |   |     |   +-parseReplacepage()
              |   |     |   | <void parseReplacepage () line:10402>
918 {   9}    |   |     |   | +-getExpectedToken()
              |   |     |   | | <void getExpectedToken () line:2406>
              |   |     |   | | (R) [see 86]
919 {   9}    |   |     |   | +-getToken()
              |   |     |   | | <int getToken () line:2535> (R) [see 74]
920 {   9}    |   |     |   | \-allocString()
              |   |     |   |   <char *allocString () line:2189> [see 61]
921 {   8}    |   |     |   \-printPageAndFilename()
              |   |     |     <void printPageAndFilename () line:2286>
              |   |     |     [see 92]
922 {   7}    |   |     | +-printToString()
              |   |     | | <char *printToString () line:13497> (R) [see 445]
923 {   7}    |   |     | +-XSetIconName()
924 {   7}    |   |     | +-fprintf()
925 {   7}    |   |     | +-printPageAndFilename()
              |   |     | |<void printPageAndFilename () line:2286> [see 92]
926 {   7}    |   |     | +-jump() <void jump () line:2196> [see 108]
927 {   7}    |   |     | +-linkTitleBarWindows()
              |   |     | | <void linkTitleBarWindows () line:14406>
928 {   8}    |   |     | | +-halloc() <char *halloc () line:2070> [see 8]
929 {   8}    |   |     | | \-hashInsert()
              |   |     | |   <void hashInsert () line:2104> [see 24]
930 {   7}    |   |     | \-PopMR() <void PopMR () line:9743> [see 558]
931 {   6}    |   |     \-parseHeader() <void parseHeader () line:9866>
932 {   7}    |   |       +-PushMR() <void PushMR () line:9734> [see 416]
933 {   7}    |   |       +-allocNode()
              |   |       | <TextNode *allocNode () line:9265> [see 419]
934 {   7}    |   |       \-parseHyperDoc()
              |   |         <void parseHyperDoc () line:9938> (R) [see 421]
935 {   3}    |   \-makeInputFileFromPage()
              |     <void makeInputFileFromPage () line:7533>
936 {   4}    |     +-makeInputFileName()
              |     | <char *makeInputFileName () line:7494> [see 341]
937 {   4}    |     +-makePasteFileName()
              |     | <char *makePasteFileName () line:7505>
938 {   5}    |     | +-strcpy()
939 {   5}    |     | \-strlen()
940 {   4}    |     +-inListAndNewer()
              |     |  <int inListAndNewer () line:7618> [see 349]
941 {   4}    |     +-fopen()
942 {   4}    |     +-fprintf()
943 {   4}    |     +-exit()
944 {   4}    |     +-sendLispCommand() <void sendLispCommand () line:13867>
945 {   5}    |     | +-connectSpad() <int connectSpad () line:1879>
946 {   6}    |     |   +-fprintf()
```

```
947 {   6}   |     |   +-LoudBeepAtTheUser()
948 {   6}   |     |   \-connect_to_local_server()
949 {   5}   |     | +-send_int()
950 {   5}   |     | \-send_string()
951 {   4}   |     +-printToString()
             |     | <char *printToString () line:13497> (R) [see 445]
952 {   4}   |     +-allocString() <char *allocString () line:2189> [see 61]
953 {   4}   |     +-fflush()
954 {   4}   |     +-printPaste() <void printPaste () line:7762>
955 {   5}   |     | +-fprintf()
956 {   5}   |     | +-printPasteLine() <void printPasteLine () line:7680>
957 {   6}   |     | | \-putc()
958 {   5}   |     | +-getSpadOutput() <void getSpadOutput () line:7708>
959 {   6}   |     | | +-sendCommand() <void sendCommand () line:7739>
960 {   7}   |     | | | +-escapeString() <void escapeString () line:13877>
961 {   8}   |     | | | | \-funnyEscape()
962 {   7}   |     | | | +-sprintf()
963 {   7}   |     | | | +-sendLispCommand()
             |     | | |   <void sendLispCommand () line:13867> [see 944]
964 {   7}   |     | | | +-getenv()
965 {   7}   |     | | | +-fopen()
966 {   7}   |     | | | +-fprintf()
967 {   7}   |     | | | \-fclose()
968 {   6}   |     | | +-get_int()
969 {   6}   |     | | +-get_string_buf()
970 {   6}   |     | | +-fprintf()
971 {   6}   |     | | \-unescapeString() <void unescapeString () line:13883>
972 {   7}   |     | |   \-funnyUnescape()
973 {   5}   |     | \-fflush()
974 {   4}   |     +-printGraphPaste() <void printGraphPaste () line:7789>
975 {   5}   |     | +-fprintf()
976 {   5}   |     | +-printPasteLine()
             |     | | <void printPasteLine () line:7680> [see 956]
977 {   5}   |     | +-getGraphOutput() <void getGraphOutput () line:7720>
978 {   6}   |     | | +-sendCommand()
             |     | | | <void sendCommand () line:7739> [see 959]
979 {   6}   |     | | +-get_int()
980 {   6}   |     | | +-get_string_buf()
981 {   6}   |     | | +-unescapeString()
             |     | | | <void unescapeString () line:13883> [see 971]
982 {   6}   |     | | +-sprintf()
983 {   6}   |     | | \-sendLispCommand()
             |     | |   <void sendLispCommand () line:13867> [see 944]
984 {   5}   |     | \-fflush()
985 {   4}   |     \-fclose()
986 {   2}   | +-connectSpad() <int connectSpad () line:1879> [see 945]
987 {   2}   | \-send_int()
988 {   1}   +-makeRecord() <void makeRecord () line:7437>
989 {   2}   | +-sendLispCommand()
             |   <void sendLispCommand () line:13867> [see 944]
```

```
 990 {   2}   | +-sprintf()
 991 {   2}   | +-fprintf()
 992 {   2}   | +-connectSpad() <int connectSpad () line:1879> [see 945]
 993 {   2}   | \-send_int()
 994 {   1}   +-verifyRecord() <void verifyRecord () line:7456>
 995 {   2}   | +-sendLispCommand()
              |   <void sendLispCommand () line:13867> [see 944]
 996 {   2}   | +-sprintf()
 997 {   2}   | +-fprintf()
 998 {   2}   | +-connectSpad() <int connectSpad () line:1879> [see 945]
 999 {   2}   | \-send_int()
1000 {   1}   \-mainEventLoop() <void mainEventLoop () line:4552>
1001 {   2}     +-setErrorHandlers() <void setErrorHandlers () line:5390>
1002 {   3}     | +-XSetErrorHandler()
1003 {   3}     | \-HyperDocErrorHandler()
              |   <int HyperDocErrorHandler () line:5375>
1004 {   4}     |   +-XGetErrorText()
1005 {   4}     |   +-fprintf()
1006 {   4}     |   \-exit()
1007 {   2}     +-ConnectionNumber()
1008 {   2}     +-pause()
1009 {   2}     +-initCursorStates() <void initCursorStates () line:5363>
1010 {   3}       +-hashMap() <void hashMap () line:2129>
1011 {   3}       \-initCursorState() <void initCursorState () line:5350>
1012 {   4}         +-XQueryPointer()
1013 {   4}         +-findButtonInList()
                   | <HyperLink *findButtonInList () line:4929>
1014 {   4}         \-changeCursor() <void changeCursor () line:5334>
1015 {   5}           \-setCursor() <void setCursor () line:5324>
1016 {   6}             +-XDefineCursor()
1017 {   6}             \-XFlush()
1018 {   2}     +-FD_ZERO()
1019 {   2}     +-FD_CLR()
1020 {   2}     +-FD_SET()
1021 {   2}     +-XEventsQueued()
1022 {   2}     +-XNextEvent()
1023 {   2}     +-handleEvent() <void handleEvent () line:4622>
1024 {   3}     | +-setWindow() <int setWindow () line:5155>
1025 {   4}     |   +-hashFind() <char *hashFind () line:2139> [see 68]
1026 {   4}     |   +-XQueryTree()
1027 {   4}     |   \-XFree()
1028 {   3}     | +-handleMotionEvent() <void handleMotionEvent () line:5341>
1029 {   4}     |   +-findButtonInList()
              |   | <HyperLink *findButtonInList () line:4929> [see 1013]
1030 {   4}     |   \-changeCursor()
              |     <void changeCursor () line:5334> [see 1014]
1031 {   3}     | +-makeBusyCursors() <void makeBusyCursors () line:5371>
1032 {   4}     |   +-hashMap() <void hashMap () line:2129> [see 1010]
1033 {   4}     |   \-makeBusyCursor() <void makeBusyCursor () line:5367>
1034 {   5}     |     \-changeCursor()
```

```
                |       <void changeCursor () line:5334> [see 1014]
1035 {   3}     | +-XGetWindowAttributes()
1036 {   3}     | +-displayPage() <void displayPage () line:9769>
1037 {   4}     |   +-XUnmapSubwindows()
1038 {   4}     |   +-XFlush()
1039 {   4}     |   +-setjmp()
1040 {   4}     |   +-freePage() <void freePage () line:9492>
1041 {   5}     |     +-freeNode() <void freeNode () line:9281> (R) [see 487]
1042 {   5}     |     +-freeButtonList() <void freeButtonList () line:9710>
1043 {   6}     |     | \-free()
1044 {   5}     |     +-freeHash() <void freeHash () line:2160> [see 141]
1045 {   5}     |     +-freeDepend() <void freeDepend () line:9593>
1046 {   6}     |     | \-freeIfNonNULL()
                |     |   <void freeIfNonNULL () line:9201> [see 144]
1047 {   5}     |     +-dontFree() <void dontFree () line:9597>
1048 {   5}     |     +-freeInputBox() <void freeInputBox () line:9629>
1049 {   6}     |     | +-freeIfNonNULL()
                |     |   <void freeIfNonNULL () line:9201> [see 144]
1050 {   6}     |     | \-free()
1051 {   5}     |     +-freeInputList()
                |       <void freeInputList () line:9620> [see 396]
1052 {   5}     |     +-freeRadioBoxes()
                |       <void freeRadioBoxes () line:9636> (R)
1053 {   6}     |     | +-freeRadioBoxes()
                |         <void freeRadioBoxes () line:9636>
                |         (recursive: see 1052) [see 1052]
1054 {   6}     |     | +-freeIfNonNULL()
                |         <void freeIfNonNULL () line:9201> [see 144]
1055 {   6}     |     | \-free()
1056 {   5}     |     \-free()
1057 {   4}     |   +-hashReplace() <char *hashReplace () line:2148> [see 364]
1058 {   4}     |   +-strcmp()
1059 {   4}     |   +-fprintf()
1060 {   4}     |   +-exit()
1061 {   4}     |   +-hashFind() <char *hashFind () line:2139> [see 68]
1062 {   4}     |   +-resetConnection() <void resetConnection () line:1897>
1063 {   5}     |     +-FD_CLR()
1064 {   5}     |     +-close()
1065 {   5}     |     \-connectSpad() <int connectSpad () line:1879> [see 945]
1066 {   4}     |   +-initScanner() <void initScanner () line:2343> [see 40]
1067 {   4}     |   +-formatPage()
                |     <HyperDocPage *formatPage () line:9805> [see 362]
1068 {   4}     |   \-showPage() <void showPage () line:4360>
1069 {   5}     |     +-initTopGroup()
                |     | <void initTopGroup () line:7392> [see 403]
1070 {   5}     |     +-XClearWindow()
1071 {   5}     |     +-freeButtonList()
                |     | <void freeButtonList () line:9710> [see 1042]
1072 {   5}     |     +-computeTitleExtent()
                |     | <void computeTitleExtent () line:6397>
```

```
1073 {   6}     |     | +-initTitleExtents()
                |     |   <void initTitleExtents () line:6850>
1074 {   7}     |     |   \-clearItemStack()
                |     |     <void clearItemStack () line:8657>
1075 {   8}     |     |     \-free()
1076 {   6}     |     | +-computeTextExtent()
                |     | | <void computeTextExtent () line:5580> (R)
1077 {   7}     |     | | +-endpastebuttonExtent()
                |     | |   <void endpastebuttonExtent () line:6132>
1078 {   8}     |     | |   +-textWidth() <int textWidth () line:6564>
1079 {   9}     |     | |     +-punctuationWidth()
                |     | |       <int punctuationWidth () line:6515>
1080 {  10}     |     | |       +-strlen()
1081 {  10}     |     | |       \-XTextWidth()
1082 {   9}     |     | |     +-widthOfDash() <int widthOfDash () line:6551>
1083 {  10}     |     | |       +-strlen()
1084 {  10}     |     | |       \-XTextWidth()
1085 {   9}     |     | |     +-verbatimWidth()
                |     | |       <int verbatimWidth () line:6543>
1086 {  10}     |     | |       +-strlen()
1087 {  10}     |     | |       \-XTextWidth()
1088 {   9}     |     | |     +-wordWidth() <int wordWidth () line:6535>
1089 {  10}     |     | |       +-strlen()
1090 {  10}     |     | |       \-XTextWidth()
1091 {   9}     |     | |     +-pushActiveGroup()
                |     | |       <void pushActiveGroup () line:7378>
1092 {  10}     |     | |       +-pushGroupStack()
                |     | |       | <void pushGroupStack () line:7312> [see 732]
1093 {  10}     |     | |       \-changeText()
                |     | |         <void changeText () line:8372> [see 317]
1094 {   9}     |     | |     +-popGroupStack()
                |     | |       <int popGroupStack () line:7294> [see 404]
1095 {   9}     |     | |     +-inputStringWidth()
                |     | |       <int inputStringWidth () line:6524>
1096 {   9}     |     | |     +-pushSpadGroup()
                |     | |       <void pushSpadGroup () line:7385>
1097 {  10}     |     | |       +-pushGroupStack()
                |     | |       | <void pushGroupStack () line:7312> [see 732]
1098 {  10}     |     | |       \-changeText()
                |     | |         <void changeText () line:8372> [see 317]
1099 {   9}     |     | |     +-atoi()
1100 {   9}     |     | |     +-pushGroupStack()
                |     | |       <void pushGroupStack () line:7312> [see 732]
1101 {   9}     |     | |     +-bfTopGroup() <void bfTopGroup () line:7359>
1102 {  10}     |     | |       +-pushGroupStack()
                |     | |       | <void pushGroupStack () line:7312> [see 732]
1103 {  10}     |     | |       \-changeText()
                |     | |         <void changeText () line:8372> [see 317]
1104 {   9}     |     | |     +-emTopGroup() <void emTopGroup () line:7334>
1105 {  10}     |     | |     | +-pushGroupStack()
```

```
                |     | |     | | <void pushGroupStack () line:7312> [see 732]
1106 {  10}     |     | |     | \-changeText()
                |     | |     |   <void changeText () line:8372> [see 317]
1107 {   9}     |     | |     +-rmTopGroup() <void rmTopGroup () line:7342>
1108 {  10}     |     | |       +-pushGroupStack()
                |     | |       | <void pushGroupStack () line:7312> [see 732]
1109 {  10}     |     | |       \-changeText()
                |     | |         <void changeText () line:8372> [see 317]
1110 {   9}     |     | |     +-insertBitmapFile()
                |     | |     | <void insertBitmapFile () line:7131>
1111 {  10}     |     | |     | +-hashFind()
                |     | |     | | <char *hashFind () line:2139> [see 68]
1112 {  10}     |     | |     | +-getenv()
1113 {  10}     |     | |     | +-HTReadBitmapFile()
                |     | |     | | <XImage *HTReadBitmapFile () line:12233>
                |     | |     | | [see 266]
1114 {  10}     |     | |     | +-halloc() <char *halloc () line:2070> [see 8]
1115 {  10}     |     | |     | +-strlen()
1116 {  10}     |     | |     | +-sprintf()
1117 {  10}     |     | |     | \-hashInsert()
                |     | |     |   <void hashInsert () line:2104> [see 24]
1118 {   9}     |     | |     \-insertPixmapFile()
                |     | |       <void insertPixmapFile () line:7165>
1119 {  10}     |     | |       +-hashFind()
                |     | |       | <char *hashFind () line:2139> [see 68]
1120 {  10}     |     | |       +-read_pixmap_file()
1121 {  10}     |     | |       +-fprintf()
1122 {  10}     |     | |       +-halloc() <char *halloc () line:2070> [see 8]
1123 {  10}     |     | |       +-strlen()
1124 {  10}     |     | |       +-sprintf()
1125 {  10}     |     | |       +-hashInsert()
                |     | |       | <void hashInsert () line:2104> [see 24]
1126 {  10}     |     | |       \-plh() <int plh () line:7208>
1127 {   8}     |     | |   +-textHeight() <int textHeight () line:6872>
1128 {   9}     |     | |     \-textHeight1() <int textHeight1 () line:6877>
1129 {  10}     |     | |       \-max()
1130 {   8}     |     | |   +-startNewline() <void startNewline () line:6487>
1131 {   9}     |     | |     \-centerNodes() <void centerNodes () line:6499>
1132 {  10}     |     | |       \-Xvalue() <int Xvalue () line:7072> (R)
1133 {  11}     |     | |         +-fprintf()
1134 {  11}     |     | |         \-Xvalue()
                |     | |           <int Xvalue () line:7072>
                |     | |           (recursive: see 1132) [see 1132]
1135 {   8}     |     | |   \-popGroupStack()
                |     | |     <int popGroupStack () line:7294> [see 404]
1136 {   7}     |     | | +-computePasteExtent()
                |     | |   <void computePasteExtent () line:6157>
1137 {   8}     |     | |   \-startNewline()
                |     | |     <void startNewline () line:6487> [see 1130]
1138 {   7}     |     | | +-startNewline()
```

```
                |     | |   <void startNewline () line:6487> [see 1130]
1139 {   7}     |     | | +-computePastebuttonExtent()
                |     | |   <void computePastebuttonExtent () line:6113>
1140 {   8}     |     | |   +-pushActiveGroup()
                |     | |   | <void pushActiveGroup () line:7378> [see 1091]
1141 {   8}     |     | |   +-textWidth()
                |     | |   | <int textWidth () line:6564> [see 1078]
1142 {   8}     |     | |   \-startNewline()
                |     | |     <void startNewline () line:6487> [see 1130]
1143 {   7}     |     | | +-computeIfcondExtent()
                |     | |   <void computeIfcondExtent () line:5954>
1144 {   8}     |     | |   +-pushGroupStack()
                |     | |   | <void pushGroupStack () line:7312> [see 732]
1145 {   8}     |     | |   +-computeTextExtent()
                |     | |   | <void computeTextExtent () line:5580>
                |     | |   | (recursive: see 1076) [see 1076]
1146 {   8}     |     | |   +-checkCondition()
                |     | |   | <int checkCondition () line:3217> (R) [see 448]
1147 {   8}     |     | |   \-popGroupStack()
                |     | |     <int popGroupStack () line:7294> [see 404]
1148 {   7}     |     | | +-endifExtent() <void endifExtent () line:5944>
1149 {   7}     |     | | +-popGroupStack()
                |     | |   <int popGroupStack () line:7294> [see 404]
1150 {   7}     |     | | +-computePunctuationExtent()
                |     | |   <void computePunctuationExtent () line:5423>
1151 {   8}     |     | |   +-strlen()
1152 {   8}     |     | |   +-XTextWidth()
1153 {   8}     |     | |   +-totalWidth() <int totalWidth () line:6748>
1154 {   9}     |     | |     +-XTextWidth()
1155 {   9}     |     | |     \-strlen()
1156 {   8}     |     | |   \-startNewline()
                |     | |     <void startNewline () line:6487> [see 1130]
1157 {   7}     |     | | +-computeSpadsrctxtExtent()
                |     | |   <void computeSpadsrctxtExtent () line:5522>
1158 {   8}     |     | |   +-strlen()
1159 {   8}     |     | |   \-startNewline()
                |     | |     <void startNewline () line:6487> [see 1130]
1160 {   7}     |     | | +-computeWordExtent()
                |     | |   <void computeWordExtent () line:5471>
1161 {   8}     |     | |   +-strlen()
1162 {   8}     |     | |   +-XTextWidth()
1163 {   8}     |     | |   +-totalWidth()
                |     | |   | <int totalWidth () line:6748> [see 1153]
1164 {   8}     |     | |   \-startNewline()
                |     | |     <void startNewline () line:6487> [see 1130]
1165 {   7}     |     | | +-computeVerbatimExtent()
                |     | |   <void computeVerbatimExtent () line:5513>
1166 {   8}     |     | |   \-strlen()
1167 {   7}     |     | | +-computeDashExtent()
                |     | |   <void computeDashExtent () line:5535>
```

```
1168 {   8}     |     | |   +-strlen()
1169 {   8}     |     | |   +-XTextWidth()
1170 {   8}     |     | |   +-totalWidth()
                 |     | |   | <int totalWidth () line:6748> [see 1153]
1171 {   8}     |     | |   \-startNewline()
                 |     | |     <void startNewline () line:6487> [see 1130]
1172 {   7}     |     | | +-atoi()
1173 {   7}     |     | | +-computeCenterExtent()
                 |     | |   <void computeCenterExtent () line:6012>
1174 {   8}     |     | |   +-startNewline()
                 |     | |   | <void startNewline () line:6487> [see 1130]
1175 {   8}     |     | |   +-centerTopGroup()
                 |     | |   | <void centerTopGroup () line:7402>
1176 {   9}     |     | |   | \-pushGroupStack()
                 |     | |   |   <void pushGroupStack () line:7312> [see 732]
1177 {   8}     |     | |   +-fprintf()
1178 {   8}     |     | |   \-exit()
1179 {   7}     |     | | +-computeBoxExtent()
                 |     | |   <void computeBoxExtent () line:6275>
1180 {   8}     |     | |   +-textWidth()
                 |     | |   | <int textWidth () line:6564> [see 1078]
1181 {   8}     |     | |   \-startNewline()
                 |     | |     <void startNewline () line:6487> [see 1130]
1182 {   7}     |     | | +-computeMboxExtent()
                 |     | |   <void computeMboxExtent () line:6263>
1183 {   8}     |     | |   +-textWidth()
                 |     | |   | <int textWidth () line:6564> [see 1078]
1184 {   8}     |     | |   \-startNewline()
                 |     | |     <void startNewline () line:6487> [see 1130]
1185 {   7}     |     | | +-computeBeginItemsExtent()
                 |     | |   <void computeBeginItemsExtent () line:5900>
1186 {   8}     |     | |   +-startNewline()
                 |     | |   | <void startNewline () line:6487> [see 1130]
1187 {   8}     |     | |   +-pushItemStack()
                 |     | |   | <void pushItemStack () line:8647>
1188 {   9}     |     | |   | \-halloc() <char *halloc () line:2070> [see 8]
1189 {   8}     |     | |   +-computeTextExtent()
                 |     | |   | <void computeTextExtent () line:5580>
                 |     | |   | (recursive: see 1076) [see 1076]
1190 {   8}     |     | |   \-textWidth()
                 |     | |     <int textWidth () line:6564> [see 1078]
1191 {   7}     |     | | +-popItemStack() <void popItemStack () line:8667>
1192 {   8}     |     | |   +-fprintf()
1193 {   8}     |     | |   \-free()
1194 {   7}     |     | | +-computeItemExtent()
                 |     | |   <void computeItemExtent () line:5931>
1195 {   8}     |     | |   \-startNewline()
                 |     | |     <void startNewline () line:6487> [see 1130]
1196 {   7}     |     | | +-computeMitemExtent()
                 |     | |   <void computeMitemExtent () line:5937>
```

```
1197 {   8}     |     | |   \-startNewline()
                |     | |     <void startNewline () line:6487> [see 1130]
1198 {   7}     |     | | +-computeButtonExtent()
                |     | |   <void computeButtonExtent () line:6059>
1199 {   8}     |     | |   +-pushActiveGroup()
                |     | |   | <void pushActiveGroup () line:7378> [see 1091]
1200 {   8}     |     | |   +-textWidth()
                |     | |   | <int textWidth () line:6564> [see 1078]
1201 {   8}     |     | |   \-startNewline()
                |     | |     <void startNewline () line:6487> [see 1130]
1202 {   7}     |     | | +-endbuttonExtent()
                |     | |   <void endbuttonExtent () line:6081>
1203 {   8}     |     | |   +-maxX() <int maxX () line:6996>
1204 {   9}     |     | |     +-max()
1205 {   9}     |     | |     +-wordWidth()
                |     | |     | <int wordWidth () line:6535> [see 1088]
1206 {   9}     |     | |     +-verbatimWidth()
                |     | |     | <int verbatimWidth () line:6543> [see 1085]
1207 {   9}     |     | |     +-punctuationWidth()
                |     | |     | <int punctuationWidth () line:6515> [see 1079]
1208 {   9}     |     | |     +-widthOfDash()
                |     | |     | <int widthOfDash () line:6551> [see 1082]
1209 {   9}     |     | |     +-atoi()
1210 {   9}     |     | |     +-pushGroupStack()
                |     | |     | <void pushGroupStack () line:7312> [see 732]
1211 {   9}     |     | |     +-bfTopGroup()
                |     | |     | <void bfTopGroup () line:7359> [see 1101]
1212 {   9}     |     | |     +-emTopGroup()
                |     | |     | <void emTopGroup () line:7334> [see 1104]
1213 {   9}     |     | |     +-rmTopGroup()
                |     | |     | <void rmTopGroup () line:7342> [see 1107]
1214 {   9}     |     | |     +-popGroupStack()
                |     | |     | <int popGroupStack () line:7294> [see 404]
1215 {   9}     |     | |     +-insertBitmapFile()
                |     | |     | <void insertBitmapFile () line:7131>
                |     | |     | [see 1110]
1216 {   9}     |     | |     \-insertPixmapFile()
                |     | |       <void insertPixmapFile () line:7165>
                |     | |       [see 1118]
1217 {   8}     |     | |   +-textWidth()
                |     | |   | <int textWidth () line:6564> [see 1078]
1218 {   8}     |     | |   +-textHeight()
                |     | |   | <int textHeight () line:6872> [see 1127]
1219 {   8}     |     | |   +-startNewline()
                |     | |   | <void startNewline () line:6487> [see 1130]
1220 {   8}     |     | |   \-popGroupStack()
                |     | |     <int popGroupStack () line:7294> [see 404]
1221 {   7}     |     | | +-computeSpadsrcExtent()
                |     | |   <void computeSpadsrcExtent () line:6192>
1222 {   8}     |     | |   +-pushSpadGroup()
```

```
                |     | |   | <void pushSpadGroup () line:7385> [see 1096]
1223 {   8}     |     | |   \-startNewline()
                |     | |     <void startNewline () line:6487> [see 1130]
1224 {   7}     |     | | +-computeSpadcommandExtent()
                |     | |   <void computeSpadcommandExtent () line:6167>
1225 {   8}     |     | |   +-pushSpadGroup()
                |     | |   | <void pushSpadGroup () line:7385> [see 1096]
1226 {   8}     |     | |   +-textWidth()
                |     | |   | <int textWidth () line:6564> [see 1078]
1227 {   8}     |     | |   \-startNewline()
                |     | |     <void startNewline () line:6487> [see 1130]
1228 {   7}     |     | | +-endSpadsrcExtent()
                |     | |   <void endSpadsrcExtent () line:6237>
1229 {   8}     |     | |   +-maxX() <int maxX () line:6996> [see 1203]
1230 {   8}     |     | |   +-textWidth()
                |     | |   | <int textWidth () line:6564> [see 1078]
1231 {   8}     |     | |   +-textHeight()
                |     | |   | <int textHeight () line:6872> [see 1127]
1232 {   8}     |     | |   +-startNewline()
                |     | |   | <void startNewline () line:6487> [see 1130]
1233 {   8}     |     | |   \-popGroupStack()
                |     | |     <int popGroupStack () line:7294> [see 404]
1234 {   7}     |     | | +-endSpadcommandExtent()
                |     | |   <void endSpadcommandExtent () line:6211>
1235 {   8}     |     | |   +-maxX() <int maxX () line:6996> [see 1203]
1236 {   8}     |     | |   +-textWidth()
                |     | |   | <int textWidth () line:6564> [see 1078]
1237 {   8}     |     | |   +-textHeight()
                |     | |   | <int textHeight () line:6872> [see 1127]
1238 {   8}     |     | |   +-startNewline()
                |     | |   | <void startNewline () line:6487> [see 1130]
1239 {   8}     |     | |   \-popGroupStack()
                |     | |     <int popGroupStack () line:7294> [see 404]
1240 {   7}     |     | | +-pushGroupStack()
                |     | |   <void pushGroupStack () line:7312> [see 732]
1241 {   7}     |     | | +-insertBitmapFile()
                |     | |   <void insertBitmapFile () line:7131> [see 1110]
1242 {   7}     |     | | +-computeImageExtent()
                |     | |   <void computeImageExtent () line:6324>
1243 {   8}     |     | |   +-startNewline()
                |     | |   |<void startNewline () line:6487> [see 1130]
1244 {   8}     |     | |   \-plh() <int plh () line:7208> [see 1126]
1245 {   7}     |     | | +-insertPixmapFile()
                |     | |   <void insertPixmapFile () line:7165> [see 1118]
1246 {   7}     |     | | +-computeTableExtent()
                |     | |   <void computeTableExtent () line:6341>
1247 {   8}     |     | |   +-textWidth()
                |     | |   | <int textWidth () line:6564> [see 1078]
1248 {   8}     |     | |   +-startNewline()
                |     | |   | <void startNewline () line:6487> [see 1130]
```

```
1249 {   8}     |     | |   \-computeTextExtent()
                |     | |     <void computeTextExtent () line:5580>
                |     | |     (recursive: see 1076) [see 1076]
1250 {   7}     |     | | +-computeBfExtent()
                |     | |   <void computeBfExtent () line:6025>
1251 {   8}     |     | |   \-bfTopGroup()
                |     | |     <void bfTopGroup () line:7359> [see 1101]
1252 {   7}     |     | | +-computeEmExtent()
                |     | |   <void computeEmExtent () line:6033>
1253 {   8}     |     | |   +-rmTopGroup()
                |     | |   | <void rmTopGroup () line:7342> [see 1107]
1254 {   8}     |     | |   \-emTopGroup()
                |     | |     <void emTopGroup () line:7334> [see 1104]
1255 {   7}     |     | | +-computeItExtent()
                |     | |   <void computeItExtent () line:6044>
1256 {   7}     |     | | +-computeRmExtent()
                |     | |   <void computeRmExtent () line:6051>
1257 {   8}     |     | |   \-rmTopGroup()
                |     | |     <void rmTopGroup () line:7342> [see 1107]
1258 {   7}     |     | | +-computeInputExtent()
                |     | |   <void computeInputExtent () line:5394>
1259 {   8}     |     | |   +-startNewline()
                |     | |   | <void startNewline () line:6487> [see 1130]
1260 {   8}     |     | |   \-plh() <int plh () line:7208> [see 1126]
1261 {   7}     |     | | +-computeIrExtent()
                |     | |   <void computeIrExtent () line:6297>
1262 {   8}     |     | |   +-startNewline()
                |     | |   | <void startNewline () line:6487> [see 1130]
1263 {   8}     |     | |   \-plh() <int plh () line:7208> [see 1126]
1264 {   7}     |     | | +-bfTopGroup()
                |     | |   <void bfTopGroup () line:7359> [see 1101]
1265 {   7}     |     | | \-fprintf()
1266 {   6}     |     | +-max()
1267 {   6}     |     | \-textHeight()
                |     |   <int textHeight () line:6872> [see 1127]
1268 {   5}     |     +-computeHeaderExtent()
                |     | <void computeHeaderExtent () line:6410>
1269 {   6}     |     | +-initExtents() <void initExtents () line:6835>
1270 {   7}     |     |   \-clearItemStack()
                |     |     <void clearItemStack () line:8657> [see 1074]
1271 {   6}     |     | +-max()
1272 {   6}     |     | +-computeTextExtent()
                |     | | <void computeTextExtent () line:5580> (R) [see 1076]
1273 {   6}     |     | \-textHeight()
                |     |   <int textHeight () line:6872> [see 1127]
1274 {   5}     |     +-computeFooterExtent()
                |     | <void computeFooterExtent () line:6437>
1275 {   6}     |     | +-initExtents()
                |     | | <void initExtents () line:6835> [see 1269]
1276 {   6}     |     | +-computeTextExtent()
```

```
                |     | | <void computeTextExtent () line:5580> (R) [see 1076]
1277 {   6}     |     | \-textHeight()
                |     |   <int textHeight () line:6872> [see 1127]
1278 {   5}     |     +-computeScrollingExtent()
                |     | <void computeScrollingExtent () line:6459>
1279 {   6}     |     | +-initExtents()
                |     | | <void initExtents () line:6835> [see 1269]
1280 {   6}     |     | \-computeTextExtent()
                |     |   <void computeTextExtent () line:5580> (R) [see 1076]
1281 {   5}     |     +-calculateScrollBarMeasures()
                |     | <void calculateScrollBarMeasures () line:12578>
1282 {   5}     |     +-getScrollBarMinimumSize()
                |     | <void getScrollBarMinimumSize () line:12771>
1283 {   5}     |     +-XConfigureWindow()
1284 {   5}     |     +-XMapWindow()
1285 {   5}     |     +-XUnmapWindow()
1286 {   5}     |     +-hideScrollBars() <void hideScrollBars () line:12764>
1287 {   6}     |     | \-XUnmapWindow()
1288 {   5}     |     +-popGroupStack()
                |     | <int popGroupStack () line:7294> [see 404]
1289 {   5}     |     +-showText() <void showText () line:12792>
1290 {   6}     |       +-visible()
1291 {   6}     |       +-strlen()
1292 {   6}     |       +-XDrawLine()
1293 {   6}     |       +-XDrawString()
1294 {   6}     |       +-above()
1295 {   6}     |       +-below()
1296 {   6}     |       +-pushGroupStack()
                |       | <void pushGroupStack () line:7312> [see 732]
1297 {   6}     |       +-ttTopGroup() <void ttTopGroup () line:7370>
1298 {   7}     |       | +-pushGroupStack()
                |       | |   <void pushGroupStack () line:7312> [see 732]
1299 {   7}     |       | \-changeText()
                |       |   <void changeText () line:8372> [see 317]
1300 {   6}     |       +-popGroupStack()
                |       | <int popGroupStack () line:7294> [see 404]
1301 {   6}     |       +-lineTopGroup() <void lineTopGroup () line:7350>
1302 {   7}     |         +-pushGroupStack()
                |         | <void pushGroupStack () line:7312> [see 732]
1303 {   7}     |         \-changeText()
                |           <void changeText () line:8372> [see 317]
1304 {   6}     |       +-XDrawRectangle()
1305 {   6}     |       +-pix_visible()
1306 {   6}     |       +-showLink() <void showLink () line:13035>
1307 {   7}     |         +-XClearArea()
1308 {   7}     |         +-allocButtonList()
                |         | <ButtonList *allocButtonList () line:9702>
1309 {   8}     |         | \-halloc() <char *halloc () line:2070> [see 8]
1310 {   7}     |         +-pushActiveGroup()
                |         | <void pushActiveGroup () line:7378> [see 1091]
```

```
1311 {   7}     |         +-trailingSpace() <int trailingSpace () line:7122>
1312 {   8}     |           \-atoi()
1313 {   7}     |         \-rmTopGroup()
                 |           <void rmTopGroup () line:7342> [see 1107]
1314 {   6}     |       +-showSpadcommand()
                 |         <void showSpadcommand () line:13175>
1315 {   7}     |         +-pushSpadGroup()
                 |         | <void pushSpadGroup () line:7385> [see 1096]
1316 {   7}     |         +-fprintf()
1317 {   7}     |         +-XConfigureWindow()
1318 {   7}     |         \-XMapWindow()
1319 {   6}     |       +-showPastebutton()
                 |         <void showPastebutton () line:13117>
1320 {   7}     |         +-pushActiveGroup()
                 |         | <void pushActiveGroup () line:7378> [see 1091]
1321 {   7}     |         +-trailingSpace()
                 |         | <int trailingSpace () line:7122> [see 1311]
1322 {   7}     |         +-fprintf()
1323 {   7}     |         +-XConfigureWindow()
1324 {   7}     |         \-XMapWindow()
1325 {   6}     |       +-showPaste() <void showPaste () line:13100>
1326 {   7}     |         +-hashFind() <char *hashFind () line:2139> [see 68]
1327 {   7}     |         +-freeGroupStack()
                 |         | <void freeGroupStack () line:7429> [see 492]
1328 {   7}     |         +-copyGroupStack()
                 |         | <GroupItem *copyGroupStack () line:7407>
1329 {   8}     |         | \-halloc() <char *halloc () line:2070> [see 8]
1330 {   7}     |         +-freeItemStack()
                 |         | <void freeItemStack () line:8703> [see 494]
1331 {   7}     |         \-copyItemStack()
                 |           <ItemStack *copyItemStack () line:8681>
1332 {   8}     |           \-halloc() <char *halloc () line:2070> [see 8]
1333 {   6}     |       +-showImage() <void showImage () line:13195>
1334 {   7}     |         +-pix_visible()
1335 {   7}     |         +-XPutImage()
1336 {   7}     |         \-fprintf()
1337 {   6}     |       +-bfTopGroup()
                 |         <void bfTopGroup () line:7359> [see 1101]
1338 {   6}     |       +-emTopGroup()
                 |         <void emTopGroup () line:7334> [see 1104]
1339 {   6}     |       +-rmTopGroup()
                 |         <void rmTopGroup () line:7342> [see 1107]
1340 {   6}     |       +-showInput() <void showInput () line:13133>
1341 {   7}     |         +-pix_visible()
1342 {   7}     |         +-XConfigureWindow()
1343 {   7}     |         +-XMapWindow()
1344 {   7}     |         +-XFlush()
1345 {   7}     |         \-drawInputsymbol()
                 |           <void drawInputsymbol () line:3503>
1346 {   8}     |           +-XClearWindow()
```

```
1347 {   8}     |           +-XTextExtents()
1348 {   8}     |           +-XDrawString()
1349 {   8}     |           +-currentItem()
                 |           | <InputItem *currentItem () line:11291> [see 509]
1350 {   8}     |           \-drawCursor() <void drawCursor () line:3571>
1351 {   9}     |             +-XTextExtents()
1352 {   9}     |             +-XFillRectangle()
1353 {   9}     |             \-XDrawString()
1354 {   6}     |       +-showSimpleBox() <void showSimpleBox () line:13154>
1355 {   7}     |         +-visible()
1356 {   7}     |         +-XConfigureWindow()
1357 {   7}     |         +-XMapWindow()
1358 {   7}     |         +-pick_box()
1359 {   7}     |         \-unpick_box()
1360 {   6}     |       +-LoudBeepAtTheUser()
1361 {   6}     |       \-fprintf()
1362 {   5}     |     +-showScrollBars() <void showScrollBars () line:12487>
1363 {   6}     |     | +-XConfigureWindow()
1364 {   6}     |     | +-XMapWindow()
1365 {   6}     |     | \-drawScroller3DEffects()
                 |     |   <void drawScroller3DEffects () line:12464>
1366 {   7}     |     |   +-XClearWindow()
1367 {   7}     |     |   +-XDrawLine()
1368 {   7}     |     |   +-XSetBackground()
1369 {   7}     |     |   \-XSetForeground()
1370 {   5}     |     +-drawScrollLines() <void drawScrollLines () line:12551>
1371 {   6}     |       +-lineTopGroup()
                 |         <void lineTopGroup () line:7350> [see 1301]
1372 {   6}     |       +-XDrawLine()
1373 {   6}     |       +-tophalf()
1374 {   6}     |       +-bothalf()
1375 {   6}     |       \-popGroupStack()
                 |         <int popGroupStack () line:7294> [see 404]
1376 {   5}     |     +-fprintf()
1377 {   5}     |     +-showTitleBar() <void showTitleBar () line:14345>
1378 {   6}     |     | +-pushActiveGroup()
                 |     |   <void pushActiveGroup () line:7378> [see 1091]
1379 {   6}     |     | +-XConfigureWindow()
1380 {   6}     |     | +-XMapWindow()
1381 {   6}     |     | +-XPutImage()
1382 {   6}     |     | +-popGroupStack()
                 |     |   <int popGroupStack () line:7294> [see 404]
1383 {   6}     |     | +-showText() <void showText () line:12792> [see 1289]
1384 {   6}     |     | +-lineTopGroup()
                 |     |   <void lineTopGroup () line:7350> [see 1301]
1385 {   6}     |     | \-XDrawLine()
1386 {   5}     |     \-XFlush()
1387 {   3}     | +-exposePage() <void exposePage () line:4446>
1388 {   4}     | | +-initTopGroup()
                 | |   <void initTopGroup () line:7392> [see 403]
```

```
1389 {   4}     | | +-showText() <void showText () line:12792> [see 1289]
1390 {   4}     | | +-getScrollBarMinimumSize()
                 | |   <void getScrollBarMinimumSize () line:12771> [see 1282]
1391 {   4}     | | +-XUnmapWindow()
1392 {   4}     | | +-hideScrollBars()
                 | |   <void hideScrollBars () line:12764> [see 1286]
1393 {   4}     | | +-showScrollBars()
                 | |   <void showScrollBars () line:12487> [see 1362]
1394 {   4}     | | +-drawScrollLines()
                 | |   <void drawScrollLines () line:12551> [see 1370]
1395 {   4}     | | +-showTitleBar()
                 | |   <void showTitleBar () line:14345> [see 1377]
1396 {   4}     | | \-XFlush()
1397 {   3}     | +-XFlush()
1398 {   3}     | +-clearExposures() <void clearExposures () line:5206>
1399 {   4}     |   +-XFlush()
1400 {   4}     |   \-XCheckTypedWindowEvent()
1401 {   3}     | +-handleButton() <void handleButton () line:4952>
1402 {   4}     |   +-scrollUp() <void scrollUp () line:12660>
1403 {   5}     |   | +-changeWindowBackgroundPixmap()
                 |   |   <void changeWindowBackgroundPixmap () line:12783>
1404 {   6}     |   |   +-XChangeWindowAttributes()
1405 {   6}     |   |   \-XClearWindow()
1406 {   5}     |   | +-XCopyArea()
1407 {   5}     |   | +-XClearArea()
1408 {   5}     |   | \-scrollPage() <void scrollPage () line:4492>
1409 {   6}     |   |   +-initTopGroup()
                 |   |   | <void initTopGroup () line:7392> [see 403]
1410 {   6}     |   |   +-freeButtonList()
                 |   |   | <void freeButtonList () line:9710> [see 1042]
1411 {   6}     |   |   +-XUnmapSubwindows()
1412 {   6}     |   |   +-showText() <void showText () line:12792> [see 1289]
1413 {   6}     |   |   +-moveScroller() <void moveScroller () line:12537>
1414 {   7}     |   |   | +-XConfigureWindow()
1415 {   7}     |   |   | \-drawScroller3DEffects()
                 |   |   |   <void drawScroller3DEffects () line:12464>
                 |   |   |   [see 1365]
1416 {   6}     |   |   \-XFlush()
1417 {   4}     |   +-scrollDown() <void scrollDown () line:12699>
1418 {   5}     |   | +-changeWindowBackgroundPixmap()
                 |   |   <void changeWindowBackgroundPixmap () line:12783>
                 |   |   [see 1403]
1419 {   5}     |   | +-XCopyArea()
1420 {   5}     |   | +-XClearArea()
1421 {   5}     |   | \-scrollPage() <void scrollPage () line:4492> [see 1408]
1422 {   4}     |   +-getHyperLink() <HyperLink *getHyperLink () line:4942>
1423 {   5}     |   | +-hashFind() <char *hashFind () line:2139> [see 68]
1424 {   5}     |   | \-findButtonInList()
                 |   |   <HyperLink *findButtonInList () line:4929> [see 1013]
1425 {   4}     |   +-pasteButton() <HyperDocPage *pasteButton () line:4890>
```

```
1426 {   5}     |     +-BeepAtTheUser()
1427 {   5}     |     +-parsePatch() <HyperDocPage *parsePatch () line:11495>
1428 {   6}     |     | +-printToString()
                |     | | <char *printToString () line:13497> (R) [see 445]
1429 {   6}     |     | +-hashFind() <char *hashFind () line:2139> [see 68]
1430 {   6}     |     | +-fprintf()
1431 {   6}     |     | +-BeepAtTheUser()
1432 {   6}     |     | +-loadPatch() <void loadPatch () line:11614>
1433 {   7}     |     |   +-saveScannerState()
                |     |   | <void saveScannerState () line:2361> [see 436]
1434 {   7}     |     |   +-findFp() <FILE *findFp () line:10878> [see 365]
1435 {   7}     |     |   +-initScanner()
                |     |   | <void initScanner () line:2343> [see 40]
1436 {   7}     |     |   +-getExpectedToken()
                |     |   | <void getExpectedToken () line:2406> (R) [see 86]
1437 {   7}     |     |   +-strcmp()
1438 {   7}     |     |   +-fprintf()
1439 {   7}     |     |   +-jump() <void jump () line:2196> [see 108]
1440 {   7}     |     |   +-scanHyperDoc()
                |     |   | <void scanHyperDoc () line:8914> [see 645]
1441 {   7}     |     |   +-fseek()
1442 {   7}     |     |   +-halloc() <char *halloc () line:2070> [see 8]
1443 {   7}     |     |   +-getc()
1444 {   7}     |     |   \-restoreScannerState()
                |     |     <void restoreScannerState () line:2377> [see 439]
1445 {   6}     |     | +-issueServerpaste()
                |     |   <int issueServerpaste () line:13787>
1446 {   7}     |     |   +-connectSpad()
                |     |   | <int connectSpad () line:1879> [see 945]
1447 {   7}     |     |   +-switchFrames() <void switchFrames () line:13853>
1448 {   8}     |     |   | +-fprintf()
1449 {   8}     |     |   | \-send_int()
1450 {   7}     |     |   +-send_int()
1451 {   7}     |     |   +-printToString()
                |     |   | <char *printToString () line:13497> (R) [see 445]
1452 {   7}     |     |   \-send_string()
1453 {   6}     |     | +-issueUnixpaste() <int issueUnixpaste () line:13827>
1454 {   7}     |     | | +-printToString()
                |     |   | <char *printToString () line:13497> (R) [see 445]
1455 {   7}     |     | | +-popen()
1456 {   7}     |     | | +-fprintf()
1457 {   7}     |     | | \-exit()
1458 {   6}     |     | +-exit()
1459 {   6}     |     | +-setjmp()
1460 {   6}     |     | +-freeNode()
                |     | | <void freeNode () line:9281> (R) [see 487]
1461 {   6}     |     | +-initParsePatch() <void initParsePatch () line:9904>
1462 {   7}     |     | | +-initTopGroup()
                |     | | | <void initTopGroup () line:7392> [see 403]
1463 {   7}     |     | | \-clearBeStack()
```

```
                |     | |   <int clearBeStack () line:2700> [see 408]
1464 {   6}     |     | +-initPasteItem() <void initPasteItem () line:11262>
1465 {   6}     |     | +-getToken() <int getToken () line:2535> (R) [see 74]
1466 {   6}     |     | +-jump() <void jump () line:2196> [see 108]
1467 {   6}     |     | +-tokenName() <void tokenName () line:2204> [see 88]
1468 {   6}     |     | +-printPageAndFilename()
                |     | | <void printPageAndFilename () line:2286> [see 92]
1469 {   6}     |     | +-allocNode()
                |     | | <TextNode *allocNode () line:9265> [see 419]
1470 {   6}     |     | +-parseHyperDoc()
                |     | | <void parseHyperDoc () line:9938> (R) [see 421]
1471 {   6}     |     | +-free()
1472 {   6}     |     | +-repasteItem() <void repasteItem () line:11275>
1473 {   7}     |     | | \-currentItem()
                |     | |   <InputItem *currentItem () line:11291> [see 509]
1474 {   6}     |     | \-pastePage() <void pastePage () line:4508>
1475 {   7}     |     |   +-freeButtonList()
                |     |   | <void freeButtonList () line:9710> [see 1042]
1476 {   7}     |     |   +-XUnmapSubwindows()
1477 {   7}     |     |   +-initTopGroup()
                |     |   | <void initTopGroup () line:7392> [see 403]
1478 {   7}     |     |   +-computeScrollingExtent()
                |     |   | <void computeScrollingExtent () line:6459>
                |     |   | [see 1278]
1479 {   7}     |     |   +-calculateScrollBarMeasures()
                |     |   | <void calculateScrollBarMeasures () line:12578>
                |     |   | [see 1281]
1480 {   7}     |     |   +-getScrollBarMinimumSize()
                |     |   | <void getScrollBarMinimumSize () line:12771>
                |     |   | [see 1282]
1481 {   7}     |     |   +-XClearArea()
1482 {   7}     |     |   +-XFlush()
1483 {   7}     |     |   +-XClearWindow()
1484 {   7}     |     |   +-showText()
                |     |   | <void showText () line:12792> [see 1289]
1485 {   7}     |     |   +-hideScrollBars()
                |     |   | <void hideScrollBars () line:12764> [see 1286]
1486 {   7}     |     |   +-XUnmapWindow()
1487 {   7}     |     |   +-showScrollBars()
                |     |   | <void showScrollBars () line:12487> [see 1362]
1488 {   7}     |     |   \-drawScrollLines()
                |     |     <void drawScrollLines () line:12551> [see 1370]
1489 {   5}     |     \-strcmp()
1490 {   4}     |   +-printToString()
                |     <char *printToString () line:13497> (R) [see 445]
1491 {   4}     |   +-hashFind() <char *hashFind () line:2139> [see 68]
1492 {   4}     |   +-helpForHyperDoc() <void helpForHyperDoc () line:4910>
1493 {   5}     |     +-strcmp()
1494 {   5}     |     +-allocString()
                |     | <char *allocString () line:2189> [see 61]
```

```
1495 {   5}     |     +-hashFind() <char *hashFind () line:2139> [see 68]
1496 {   5}     |     +-makeWindowLink() <void makeWindowLink () line:4872>
1497 {   6}     |     | \-initTopWindow()
                |     |   <int initTopWindow () line:7871> [see 214]
1498 {   5}     |     \-BeepAtTheUser()
1499 {   4}     |   +-scrollScroller() <void scrollScroller () line:12734>
1500 {   5}     |     +-XClearWindow()
1501 {   5}     |     \-scrollPage() <void scrollPage () line:4492> [see 1408]
1502 {   4}     |   +-changeInputFocus() <void changeInputFocus () line:8538>
1503 {   5}     |     +-currentItem()
                |     | <InputItem *currentItem () line:11291> [see 509]
1504 {   5}     |     +-XConfigureWindow()
1505 {   5}     |     \-updateInputsymbol()
                |       <void updateInputsymbol () line:3534>
1506 {   6}     |       +-XTextExtents()
1507 {   6}     |       +-XClearArea()
1508 {   6}     |       +-XDrawString()
1509 {   6}     |       \-drawCursor()
                |         <void drawCursor () line:3571> [see 1350]
1510 {   4}     |   +-XConvertSelection()
1511 {   4}     |   +-XInternAtom()
1512 {   4}     |   +-toggleInputBox() <void toggleInputBox () line:8499>
1513 {   5}     |     +-unpick_box()
1514 {   5}     |     \-pick_box()
1515 {   4}     |   +-toggleRadioBox() <void toggleRadioBox () line:8512>
1516 {   5}     |     +-clearRbs() <void clearRbs () line:8528>
1517 {   6}     |     | \-unpick_box()
1518 {   5}     |     \-pick_box()
1519 {   4}     |   +-quitHyperDoc() <void quitHyperDoc () line:4750>
1520 {   5}     |     +-strcmp()
1521 {   5}     |     +-exitHyperDoc() <void exitHyperDoc () line:5122>
1522 {   6}     |     | +-freeHdWindow() <void freeHdWindow () line:9239>
1523 {   7}     |     | | +-free()
1524 {   7}     |     | | +-freeHash() <void freeHash () line:2160> [see 141]
1525 {   7}     |     | | +-dontFree() <void dontFree () line:9597>
1526 {   7}     |     | | +-freeCond() <void freeCond () line:9463>
1527 {   8}     |     | | | \-free()
1528 {   7}     |     | | +-freePage() <void freePage () line:9492> [see 1040]
1529 {   7}     |     | | \-XFreeGC()
1530 {   6}     |     | +-exit()
1531 {   6}     |     | +-hashDelete()
                |     | | <void hashDelete () line:2118> [see 490]
1532 {   6}     |     | +-XFlush()
1533 {   6}     |     | +-XCheckWindowEvent()
1534 {   6}     |     | \-XDestroyWindow()
1535 {   5}     |     +-hashFind() <char *hashFind () line:2139> [see 68]
1536 {   5}     |     +-fprintf()
1537 {   5}     |     \-displayPage()
                |       <void displayPage () line:9769> [see 1036]
1538 {   4}     |   +-returnlink() <HyperDocPage *returnlink () line:4831>
```

```
1539 {   5}     |     +-BeepAtTheUser()
1540 {   5}     |     \-killPage() <void killPage () line:4822>
1541 {   6}     |       +-hashDelete()
                |       | <void hashDelete () line:2118> [see 490]
1542 {   6}     |       +-killAxiomPage() <void killAxiomPage () line:4816>
1543 {   7}     |       | +-sprintf()
1544 {   7}     |       | \-sendLispCommand()
                |       |   <void sendLispCommand () line:13867> [see 944]
1545 {   6}     |       \-freePage() <void freePage () line:9492> [see 1040]
1546 {   4}     |   +-uplink() <HyperDocPage *uplink () line:4853>
1547 {   5}     |     +-returnlink()
                |     | <HyperDocPage *returnlink () line:4831> [see 1538]
1548 {   5}     |     \-killPage() <void killPage () line:4822> [see 1540]
1549 {   4}     |   +-findPage() <HyperDocPage *findPage () line:4776>
1550 {   5}     |   | +-printToString()
                |     | <char *printToString () line:13497> (R) [see 445]
1551 {   5}     |   | +-hashFind() <char *hashFind () line:2139> [see 68]
1552 {   5}     |   | \-fprintf()
1553 {   4}     |   +-NotSpecial()
1554 {   4}     |   +-downlink() <void downlink () line:4799>
1555 {   5}     |     \-fprintf()
1556 {   4}     |   +-memolink() <void memolink () line:4806>
1557 {   5}     |     \-fprintf()
1558 {   4}     |   +-windowlinkHandler()
                |     <void windowlinkHandler () line:4862>
1559 {   5}     |     +-printToString()
                |     | <char *printToString () line:13497> (R) [see 445]
1560 {   5}     |     \-initTopWindow()
                |       <int initTopWindow () line:7871> [see 214]
1561 {   4}     |   +-lispwindowlinkHandler()
                |     <void lispwindowlinkHandler () line:4878>
1562 {   5}     |     +-initTopWindow()
                |     | <int initTopWindow () line:7871> [see 214]
1563 {   5}     |     \-issueServerCommand()
                |       <HyperDocPage *issueServerCommand () line:13740>
1564 {   6}     |       +-connectSpad()
                |       | <int connectSpad () line:1879> [see 945]
1565 {   6}     |       +-hashFind() <char *hashFind () line:2139> [see 68]
1566 {   6}     |       +-fprintf()
1567 {   6}     |       +-switchFrames()
                |       | <void switchFrames () line:13853> [see 1447]
1568 {   6}     |       +-send_int()
1569 {   6}     |       +-printToString()
                |       | <char *printToString () line:13497> (R) [see 445]
1570 {   6}     |       +-send_string()
1571 {   6}     |       \-parsePageFromSocket()
                |         <HyperDocPage *parsePageFromSocket () line:10261>
1572 {   7}     |         +-allocPage()
                |         | <HyperDocPage *allocPage () line:9472> [see 229]
1573 {   7}     |         +-initScanner()
```

```
                |         | <void initScanner () line:2343> [see 40]
1574 {   7}     |         +-hashInit() <void hashInit () line:2091> [see 7]
1575 {   7}     |         +-windowEqual() <int windowEqual () line:10409>
1576 {   7}     |         +-windowCode() <int windowCode () line:10413>
1577 {   7}     |         +-setjmp()
1578 {   7}     |         +-freePage() <void freePage () line:9492> [see 1040]
1579 {   7}     |         +-hashFind() <char *hashFind () line:2139> [see 68]
1580 {   7}     |         +-resetConnection()
                |         | <void resetConnection () line:1897> [see 1062]
1581 {   7}     |         +-parsePage()
                |         | <void parsePage () line:9917> [see 394]
1582 {   7}     |         +-hashReplace()
                |         | <char *hashReplace () line:2148> [see 364]
1583 {   7}     |         +-hashInsert()
                |         | <void hashInsert () line:2104> [see 24]
1584 {   7}     |         \-fprintf()
1585 {   4}     |   +-issueServerCommand()
                |   | <HyperDocPage *issueServerCommand () line:13740>
                |   | [see 1563]
1586 {   4}     |   +-exitHyperDoc()
                |   | <void exitHyperDoc () line:5122> [see 1521]
1587 {   4}     |   +-issueSpadcommand()
                |     <void issueSpadcommand () line:13251> (R)
1588 {   5}     |     +-connectSpad() <int connectSpad () line:1879> [see 945]
1589 {   5}     |     +-startUserBuffer() <void startUserBuffer () line:13348>
1590 {   6}     |     | +-getenv()
1591 {   6}     |     | +-sprintf()
1592 {   6}     |     | +-printToString()
                |     | | <char *printToString () line:13497> (R) [see 445]
1593 {   6}     |     | +-access()
1594 {   6}     |     | +-system()
1595 {   6}     |     | +-acceptMenuServerConnection()
                |     | | <void acceptMenuServerConnection () line:13446>
1596 {   7}     |     | | +-sselect()
1597 {   7}     |     | | +-perror()
1598 {   7}     |     | | +-FD_ISSET()
1599 {   7}     |     | | +-acceptMenuConnection()
                |     | |   <Sock *acceptMenuConnection () line:13416>
1600 {   8}     |     | |   +-halloc() <char *halloc () line:2070> [see 8]
1601 {   8}     |     | |   +-accept()
1602 {   8}     |     | |   +-perror()
1603 {   8}     |     | |   +-get_socket_type()
1604 {   8}     |     | |   +-fprintf()
1605 {   8}     |     | |   \-FD_SET()
1606 {   7}     |     | | +-get_string()
1607 {   7}     |     | | +-hashFind() <char *hashFind () line:2139> [see 68]
1608 {   7}     |     | | \-strcmp()
1609 {   6}     |     | \-sleep()
1610 {   5}     |     +-send_int()
1611 {   5}     |     +-clearExecutionMarks()
```

```
                |     | <void clearExecutionMarks () line:13403>
1612 {   5}     |     +-issueSpadcommand()
                |     | <void issueSpadcommand () line:13251>
                |     | (recursive: see 1587) [see 1587]
1613 {   5}     |     +-issueDependentCommands()
                |     | <void issueDependentCommands () line:13297>
1614 {   6}     |     | +-hashFind() <char *hashFind () line:2139> [see 68]
1615 {   6}     |     | +-fprintf()
1616 {   6}     |     | +-issueSpadcommand()
                |     | | <void issueSpadcommand () line:13251>
                |     | | (recursive: see 1587) [see 1587]
1617 {   6}     |     | +-pause()
1618 {   6}     |     | \-sleep()
1619 {   5}     |     +-printToString()
                |     | <char *printToString () line:13497> (R) [see 445]
1620 {   5}     |     +-strlen()
1621 {   5}     |     +-sendPile() <void sendPile () line:13282>
1622 {   6}     |     | +-sprintf()
1623 {   6}     |     | +-getenv()
1624 {   6}     |     | +-fopen()
1625 {   6}     |     | +-fprintf()
1626 {   6}     |     | +-fclose()
1627 {   6}     |     | \-send_string()
1628 {   5}     |     +-send_string()
1629 {   5}     |     \-markAsExecuted() <void markAsExecuted () line:13327>
1630 {   6}     |       +-hashFind() <char *hashFind () line:2139> [see 68]
1631 {   6}     |       \-fprintf()
1632 {   4}     |   +-issueUnixlink()
                |     <HyperDocPage *issueUnixlink () line:13813>
1633 {   5}     |     +-printToString()
                |     | <char *printToString () line:13497> (R) [see 445]
1634 {   5}     |     +-popen()
1635 {   5}     |     +-fprintf()
1636 {   5}     |     +-exit()
1637 {   5}     |     +-bsdSignal()
1638 {   5}     |     +-parsePageFromUnixfd()
                |     | <HyperDocPage *parsePageFromUnixfd () line:10303>
1639 {   6}     |     | +-allocPage()
                |     | |   <HyperDocPage *allocPage () line:9472> [see 229]
1640 {   6}     |     | +-initScanner()
                |     | |   <void initScanner () line:2343> [see 40]
1641 {   6}     |     | +-hashInit() <void hashInit () line:2091> [see 7]
1642 {   6}     |     | +-windowEqual() <int windowEqual () line:10409>
1643 {   6}     |     | +-windowCode() <int windowCode () line:10413>
1644 {   6}     |     | +-setjmp()
1645 {   6}     |     | +-freePage() <void freePage () line:9492> [see 1040]
1646 {   6}     |     | +-hashFind() <char *hashFind () line:2139> [see 68]
1647 {   6}     |     | +-resetConnection()
                |     | |   <void resetConnection () line:1897> [see 1062]
1648 {   6}     |     | \-parsePage() <void parsePage () line:9917> [see 394]
```

```
1649 {   5}     |     \-sigusr2Handler()
                |       <void sigusr2Handler () line:2998> [see 324]
1650 {   4}     |   +-issueUnixcommand() <void issueUnixcommand () line:13800>
1651 {   5}     |     +-printToString()
                |     | <char *printToString () line:13497> (R) [see 445]
1652 {   5}     |     +-halloc() <char *halloc () line:2070> [see 8]
1653 {   5}     |     +-strlen()
1654 {   5}     |     +-strcpy()
1655 {   5}     |     +-system()
1656 {   5}     |     \-free()
1657 {   4}     |   \-displayPage() <void displayPage () line:9769> [see 1036]
1658 {   3}     | +-XCheckTypedWindowEvent()
1659 {   3}     | +-handleKey() <void handleKey () line:8713>
1660 {   4}     |   +-XLookupString()
1661 {   4}     |   +-scrollUpPage() <void scrollUpPage () line:12678>
1662 {   5}     |   | +-ch() <int ch () line:12776> [see 286]
1663 {   5}     |   | +-XClearWindow()
1664 {   5}     |   | \-scrollPage() <void scrollPage () line:4492> [see 1408]
1665 {   4}     |   +-scrollDownPage() <void scrollDownPage () line:12718>
1666 {   5}     |   | +-ch() <int ch () line:12776> [see 286]
1667 {   5}     |   | +-XClearWindow()
1668 {   5}     |   | \-scrollPage() <void scrollPage () line:4492> [see 1408]
1669 {   4}     |   +-quitHyperDoc()
                |   | <void quitHyperDoc () line:4750> [see 1519]
1670 {   4}     |   +-sprintf()
1671 {   4}     |   +-system()
1672 {   4}     |   +-hashFind() <char *hashFind () line:2139> [see 68]
1673 {   4}     |   +-fclose()
1674 {   4}     |   +-hashDelete() <void hashDelete () line:2118> [see 490]
1675 {   4}     |   +-halloc() <char *halloc () line:2070> [see 8]
1676 {   4}     |   +-hashInit() <void hashInit () line:2091> [see 7]
1677 {   4}     |   +-stringEqual() <int stringEqual () line:2185> [see 14]
1678 {   4}     |   +-stringHash() <int stringHash () line:2177>
1679 {   4}     |   +-makeSpecialPages()
                |   | <void makeSpecialPages () line:10720> [see 226]
1680 {   4}     |   +-readHtDb() <void readHtDb () line:10428> [see 27]
1681 {   4}     |   +-fprintf()
1682 {   4}     |   +-exit()
1683 {   4}     |   +-makeWindowLink()
                |   | <void makeWindowLink () line:4872> [see 1496]
1684 {   4}     |   +-prevInputFocus() <void prevInputFocus () line:8583>
1685 {   5}     |     +-currentItem()
                |     | <InputItem *currentItem () line:11291> [see 509]
1686 {   5}     |     +-BeepAtTheUser()
1687 {   5}     |     \-drawInputsymbol()
                |       <void drawInputsymbol () line:3503> [see 1345]
1688 {   4}     |   +-BeepAtTheUser()
1689 {   4}     |   +-nextInputFocus() <void nextInputFocus () line:8561>
1690 {   5}     |     +-currentItem()
                |     | <InputItem *currentItem () line:11291> [see 509]
```

```
1691 {   5}     |     +-BeepAtTheUser()
1692 {   5}     |     \-drawInputsymbol()
                |       <void drawInputsymbol () line:3503> [see 1345]
1693 {   4}     |   +-currentItem()
                |   | <InputItem *currentItem () line:11291> [see 509]
1694 {   4}     |   +-allocString() <char *allocString () line:2189> [see 61]
1695 {   4}     |   +-helpForHyperDoc()
                |   | <void helpForHyperDoc () line:4910> [see 1492]
1696 {   4}     |   +-scrollToFirstPage()
                |   | <void scrollToFirstPage () line:12690>
1697 {   5}     |     +-XClearWindow()
1698 {   5}     |     \-scrollPage() <void scrollPage () line:4492> [see 1408]
1699 {   4}     |   +-scrollUp() <void scrollUp () line:12660> [see 1402]
1700 {   4}     |   +-scrollDown() <void scrollDown () line:12699> [see 1417]
1701 {   4}     |   +-dialog() <void dialog () line:4239>
1702 {   5}     |   | +-currentItem()
                |   | | <InputItem *currentItem () line:11291> [see 509]
1703 {   5}     |   | +-BeepAtTheUser()
1704 {   5}     |   | +-enterNewLine() <void enterNewLine () line:4166>
1705 {   6}     |   |   +-allocInputline()
                |   |   | <LineStruct *allocInputline () line:9644> [see 819]
1706 {   6}     |   |   +-toughEnter() <void toughEnter () line:4100>
1707 {   7}     |   |   | +-allocInputline()
                |   |   | | <LineStruct *allocInputline () line:9644>
                |   |   | | [see 819]
1708 {   7}     |   |   | \-incLineNumbers()
                |   |   |   <void incLineNumbers () line:3265>
1709 {   6}     |   |   +-strncpy()
1710 {   6}     |   |   \-redrawWin() <void redrawWin () line:3251>
1711 {   7}     |   |     +-XUnmapSubwindows()
1712 {   7}     |   |     +-XFlush()
1713 {   7}     |   |     \-showPage() <void showPage () line:4360> [see 1068]
1714 {   5}     |   | +-addBufferToSym() <void addBufferToSym () line:3496>
1715 {   6}     |   |   +-insertBuffer() <void insertBuffer () line:3409>
1716 {   7}     |   |   | +-mystrncpy() <char *mystrncpy () line:3258>
1717 {   7}     |   |   | +-clearCursorline()
                |   |   | | <void clearCursorline () line:3393>
1718 {   8}     |   |   | | +-XTextExtents()
1719 {   8}     |   |   | | +-XClearArea()
1720 {   8}     |   |   | | \-XDrawString()
1721 {   7}     |   |   | +-drawCursor()
                |   |   | | <void drawCursor () line:3571> [see 1350]
1722 {   7}     |   |   | +-moveSymForward()
                |   |   | | <int moveSymForward () line:3350> (R)
1723 {   8}     |   |   | | +-moveSymForward()
                |   |   | | | <int moveSymForward () line:3350>
                |   |   | | | (recursive: see 1722) [see 1722]
1724 {   8}     |   |   | | +-strncpy()
1725 {   8}     |   |   | | +-allocInputline()
                |   |   | | | <LineStruct *allocInputline () line:9644>
```

```
                |   |   | | | [see 819]
1726 {   8}     |   |   | | \-incLineNumbers()
                |   |   | |   <void incLineNumbers () line:3265> [see 1708]
1727 {   7}     |   |   | +-strncpy()
1728 {   7}     |   |   | +-allocInputline()
                |   |   | | <LineStruct *allocInputline () line:9644>
                |   |   | | [see 819]
1729 {   7}     |   |   | +-incLineNumbers()
                |   |   | | <void incLineNumbers () line:3265> [see 1708]
1730 {   7}     |   |   | +-redrawWin()
                |   |   | | <void redrawWin () line:3251> [see 1710]
1731 {   7}     |   |   | \-updateInputsymbol()
                |   |   |   <void updateInputsymbol () line:3534> [see 1505]
1732 {   6}     |   |   \-overwriteBuffer()
                |   |     <void overwriteBuffer () line:3281>
1733 {   7}     |   |     +-clearCursor() <void clearCursor () line:3692>
1734 {   8}     |   |     | +-XTextExtents()
1735 {   8}     |   |     | +-XClearArea()
1736 {   8}     |   |     | \-XDrawString()
1737 {   7}     |   |     +-allocInputline()
                |   |     | <LineStruct *allocInputline () line:9644>
                |   |     | [see 819]
1738 {   7}     |   |     +-incLineNumbers()
                |   |     | <void incLineNumbers () line:3265> [see 1708]
1739 {   7}     |   |     +-XDrawString()
1740 {   7}     |   |     +-drawCursor()
                |   |     | <void drawCursor () line:3571> [see 1350]
1741 {   7}     |   |     \-redrawWin()
                |   |       <void redrawWin () line:3251> [see 1710]
1742 {   5}     |   | +-strlen()
1743 {   5}     |   | +-moveCursorHome() <void moveCursorHome () line:3601>
1744 {   6}     |   | | +-clearCursor()
                |   |     <void clearCursor () line:3692> [see 1733]
1745 {   6}     |   | | \-drawCursor()
                |   |     <void drawCursor () line:3571> [see 1350]
1746 {   5}     |   | +-deleteRestOfLine()
                |   |   <void deleteRestOfLine () line:3763>
1747 {   6}     |   |   +-decLineNumbers() <void decLineNumbers () line:3270>
1748 {   6}     |   |   +-free()
1749 {   6}     |   |   +-redrawWin() <void redrawWin () line:3251> [see 1710]
1750 {   6}     |   |   +-BeepAtTheUser()
1751 {   6}     |   |   +-decreaseLineNumbers()
                |   |   | <void decreaseLineNumbers () line:3276>
1752 {   6}     |   |   \-updateInputsymbol()
                |   |     <void updateInputsymbol () line:3534> [see 1505]
1753 {   5}     |   | +-allocString()
                |   |   <char *allocString () line:2189> [see 61]
1754 {   5}     |   | +-helpForHyperDoc()
                |   |   <void helpForHyperDoc () line:4910> [see 1492]
1755 {   5}     |   | +-moveCursorUp() <void moveCursorUp () line:3670>
```

```
1756 {   6}     |   |   +-BeepAtTheUser()
1757 {   6}     |   |   +-clearCursor()
                |   |   | <void clearCursor () line:3692> [see 1733]
1758 {   6}     |   |   \-drawCursor()
                |   |     <void drawCursor () line:3571> [see 1350]
1759 {   5}     |   | +-moveCursorDown() <void moveCursorDown () line:3650>
1760 {   6}     |   |   +-BeepAtTheUser()
1761 {   6}     |   |   +-clearCursor()
                |   |   | <void clearCursor () line:3692> [see 1733]
1762 {   6}     |   |   \-drawCursor()
                |   |     <void drawCursor () line:3571> [see 1350]
1763 {   5}     |   | +-deleteChar() <void deleteChar () line:4095>
1764 {   6}     |   |   +-deleteOneChar() <int deleteOneChar () line:4046>
1765 {   7}     |   |   | +-moveRestBack() <char moveRestBack () line:3733>
1766 {   8}     |   |   | | \-strncpy()
1767 {   7}     |   |   | +-BeepAtTheUser()
1768 {   7}     |   |   | +-deleteEoln() <void deleteEoln () line:3984>
1769 {   8}     |   |   | | +-decLineNumbers()
                |   |   | | | <void decLineNumbers () line:3270> [see 1747]
1770 {   8}     |   |   | | +-free()
1771 {   8}     |   |   | | +-redrawWin()
                |   |   | | | <void redrawWin () line:3251> [see 1710]
1772 {   8}     |   |   | | \-updateInputsymbol()
                |   |   | |   <void updateInputsymbol () line:3534> [see 1505]
1773 {   7}     |   |   | \-strncpy()
1774 {   6}     |   |   \-updateInputsymbol()
                |   |     <void updateInputsymbol () line:3534> [see 1505]
1775 {   5}     |   | +-backOverChar() <void backOverChar () line:3979>
1776 {   6}     |   |   +-moveBackOneChar() <int moveBackOneChar () line:3891>
1777 {   7}     |   |   | +-moveRestBack()
                |   |   | | <char moveRestBack () line:3733> [see 1765]
1778 {   7}     |   |   | +-BeepAtTheUser()
1779 {   7}     |   |   | +-backOverEoln() <void backOverEoln () line:3823>
1780 {   8}     |   |   | | +-decLineNumbers()
                |   |   | | | <void decLineNumbers () line:3270> [see 1747]
1781 {   8}     |   |   | | +-free()
1782 {   8}     |   |   | | +-redrawWin()
                |   |   | | | <void redrawWin () line:3251> [see 1710]
1783 {   8}     |   |   | | \-updateInputsymbol()
                |   |   | |   <void updateInputsymbol () line:3534> [see 1505]
1784 {   7}     |   |   | +-strncpy()
1785 {   7}     |   |   | +-decLineNumbers()
                |   |   | | <void decLineNumbers () line:3270> [see 1747]
1786 {   7}     |   |   | +-free()
1787 {   7}     |   |   | +-redrawWin()
                |   |   | | <void redrawWin () line:3251> [see 1710]
1788 {   7}     |   |   | \-updateInputsymbol()
                |   |   |   <void updateInputsymbol () line:3534> [see 1505]
1789 {   6}     |   |   \-updateInputsymbol()
                |   |     <void updateInputsymbol () line:3534> [see 1505]
```

```
1790 {   5}     |   | +-moveCursorBackward()
                |   |   <void moveCursorBackward () line:3709>
1791 {   6}     |   |   +-BeepAtTheUser()
1792 {   6}     |   |   +-clearCursor()
                |   |   | <void clearCursor () line:3692> [see 1733]
1793 {   6}     |   |   \-drawCursor()
                |   |     <void drawCursor () line:3571> [see 1350]
1794 {   5}     |   | +-moveCursorForward()
                |   |   <void moveCursorForward () line:3623>
1795 {   6}     |   |   +-BeepAtTheUser()
1796 {   6}     |   |   +-clearCursor()
                |   |   | <void clearCursor () line:3692> [see 1733]
1797 {   6}     |   |   \-drawCursor()
                |   |    <void drawCursor () line:3571> [see 1350]
1798 {   5}     |   | +-updateInputsymbol()
                |   |   <void updateInputsymbol () line:3534> [see 1505]
1799 {   5}     |   | \-moveCursorEnd() <void moveCursorEnd () line:3612>
1800 {   6}     |   |   +-clearCursor()
                |   |   | <void clearCursor () line:3692> [see 1733]
1801 {   6}     |   |   \-drawCursor()
                |   |     <void drawCursor () line:3571> [see 1350]
1802 {   4}     |   +-XFlush()
1803 {   4}     |   \-displayPage() <void displayPage () line:9769> [see 1036]
1804 {   3}     | +-createWindow() <void createWindow () line:4733>
1805 {   4}     |   +-XGetWindowAttributes()
1806 {   4}     |   +-displayPage() <void displayPage () line:9769> [see 1036]
1807 {   4}     |   \-XSelectInput()
1808 {   3}     | +-XInternAtom()
1809 {   3}     | +-XGetWindowProperty()
1810 {   3}     | +-addBufferToSym()
                |   <void addBufferToSym () line:3496> [see 1714]
1811 {   3}     | \-XFree()
1812 {   2}     +-select()
1813 {   2}     +-FD_ISSET()
1814 {   2}     +-get_int()
1815 {   2}     +-setWindow() <int setWindow () line:5155> [see 1024]
1816 {   2}     +-makeBusyCursors()
                | <void makeBusyCursors () line:5371> [see 1031]
1817 {   2}     +-getNewWindow() <void getNewWindow () line:5212>
1818 {   3}     | +-get_int()
1819 {   3}     | +-initTopWindow() <int initTopWindow () line:7871> [see 214]
1820 {   3}     | +-initScanner() <void initScanner () line:2343> [see 40]
1821 {   3}     | +-parsePageFromSocket()
                |   <HyperDocPage *parsePageFromSocket () line:10261>
                |   [see 1571]
1822 {   3}     | +-XFlush()
1823 {   3}     | +-get_string_buf()
1824 {   3}     | +-fprintf()
1825 {   3}     | +-initFormWindow() <int initFormWindow () line:7969>
1826 {   4}     | | +-allocHdWindow() <HDWindow *allocHdWindow () line:9207>
```

```
                | | | [see 215]
1827 {   4}     | | +-openFormWindow() <void openFormWindow () line:7913>
1828 {   5}     | | | +-strcpy()
1829 {   5}     | | | +-XrmGetResource()
1830 {   5}     | | | +-strncpy()
1831 {   5}     | | | +-XGeometry()
1832 {   5}     | | | +-getBorderProperties()
                | | | | <int getBorderProperties () line:8023> [see 244]
1833 {   5}     | | | +-XCreateSimpleWindow()
1834 {   5}     | | | +-RootWindow()
1835 {   5}     | | | +-WhitePixel()
1836 {   5}     | | | +-BlackPixel()
1837 {   5}     | | | +-makeScrollBarWindows()
                | | | | <void makeScrollBarWindows () line:12390> [see 254]
1838 {   5}     | | | +-makeTitleBarWindows()
                | | | | <void makeTitleBarWindows () line:14315> [see 262]
1839 {   5}     | | | +-setNameAndIcon()
                | | | | <void setNameAndIcon () line:7996> [see 285]
1840 {   5}     | | | +-XSelectInput()
1841 {   5}     | | | +-XDefineCursor()
1842 {   5}     | | | +-XSetNormalHints()
1843 {   5}     | | | \-XFlush()
1844 {   4}     | | +-windowWidth() <int windowWidth () line:7228>
1845 {   4}     | | +-allocPage()
                | | | <HyperDocPage *allocPage () line:9472> [see 229]
1846 {   4}     | | +-hashFind() <char *hashFind () line:2139> [see 68]
1847 {   4}     | | +-fprintf()
1848 {   4}     | | +-getGCs() <void getGCs () line:8161> [see 307]
1849 {   4}     | | +-hashInsert() <void hashInsert () line:2104> [see 24]
1850 {   4}     | | \-XChangeWindowAttributes()
1851 {   3}     | +-send_int()
1852 {   3}     | +-computeFormPage() <void computeFormPage () line:7217>
1853 {   4}     | | +-popGroupStack()
                | | | <int popGroupStack () line:7294> [see 404]
1854 {   4}     | | +-formHeaderExtent() <void formHeaderExtent () line:7240>
1855 {   5}     | | | +-initExtents()
                | | | | <void initExtents () line:6835> [see 1269]
1856 {   5}     | | | \-computeTextExtent()
                | | |   <void computeTextExtent () line:5580> (R) [see 1076]
1857 {   4}     | | +-formFooterExtent() <void formFooterExtent () line:7256>
1858 {   5}     | | | +-initExtents()
                | | | | <void initExtents () line:6835> [see 1269]
1859 {   5}     | | | +-computeTextExtent()
                | | | | <void computeTextExtent () line:5580> (R) [see 1076]
1860 {   5}     | | | \-textHeight() <int textHeight () line:6872> [see 1127]
1861 {   4}     | | +-formScrollingExtent()
                | | | <void formScrollingExtent () line:7273>
1862 {   5}     | | | +-initExtents()
                | | | | <void initExtents () line:6835> [see 1269]
1863 {   5}     | | | \-computeTextExtent()
```

```
                | | |   <void computeTextExtent () line:5580> (R) [see 1076]
1864 {   4}     | | \-windowHeight() <int windowHeight () line:7232>
1865 {   3}     | +-XMapWindow()
1866 {   3}     | +-loadPage() <void loadPage () line:9758> [see 360]
1867 {   3}     | +-hashFind() <char *hashFind () line:2139> [see 68]
1868 {   3}     | +-displayPage() <void displayPage () line:9769> [see 1036]
1869 {   3}     | +-clearExposures()
                | | <void clearExposures () line:5206> [see 1398]
1870 {   3}     | +-setWindow() <int setWindow () line:5155> [see 1024]
1871 {   3}     | \-exitHyperDoc() <void exitHyperDoc () line:5122> [see 1521]
1872 {   2}     \-serviceSessionSocket()
                  <void serviceSessionSocket () line:13837>
1873 {   3}       +-get_int()
1874 {   3}       +-closeClient() <void closeClient () line:13889>
1875 {   4}       | \-free()
1876 {   3}       \-fprintf()
```

Chapter 4

include

This chapter contains the include files. Unlike normal C programs we replace the 'include' directive by the actual chunk from this chapter. The results are that the include files never get written to disk and the compiler never sees the include directive, resulting in faster compile times.

The include files are collected from both the view* programs and the low level C code that supports them, including libspad which, in general, have the file extension of 'h1' rather than 'h'.

4.1 include/actions.h

— **include/actions.h** —

```
/* from bookvol7 chunk include/actions.h */
#define makeAViewport -1


/* Viewport Types */
#define view3DType     1
#define viewGraphType  2
#define view2DType     3
#define viewTubeType   4

/* 2D Viewport */

#define translate2D   0
#define scale2D       1
#define pointsOnOff   2
#define connectOnOff  3
#define spline2D      4
```

```
#define reset2D       5
#define hideControl2D 6
#define closeAll2D    7
#define axesOnOff2D   8
#define unitsOnOff2D  9
#define pick2D        10
#define drop2D        11
#define clear2D       12
#define ps2D          13
#define graph1        14
#define graph2        15
#define graph3        16
#define graph4        17
#define graph5        18
#define graph6        19
#define graph7        20
#define graph8        21
#define graph9        22
#define graphSelect1  23
#define graphSelect2  24
#define graphSelect3  25
#define graphSelect4  26
#define graphSelect5  27
#define graphSelect6  28
#define graphSelect7  29
#define graphSelect8  30
#define graphSelect9  31
#define query2D       32
#define zoom2Dx       33
#define zoom2Dy       34
#define translate2Dx  35
#define translate2Dy  36

#define maxButtons2D 37

#define graphStart    14  /* the index of graph1 */
#define graphSelectStart (graphStart+maxGraphs)

/* 3D Viewport */

#define controlButtonsStart3D 0

#define rotate       0
#define zoom         1
#define translate    2
#define render       3
#define hideControl  4
#define closeAll     5
#define axesOnOff    6
#define opaqueMesh   7
```

```
#define resetView    8
#define transparent  9

#define lighting     10
#define viewVolume   11
#define region3D     12
#define outlineOnOff 13

#define zoomx        14
#define zoomy        15
#define zoomz        16
#define originr      17
#define objectr      18
#define xy           19
#define xz           20
#define yz           21
#define smooth       22
#define saveit       23
#define bwColor      24

#define maxControlButtons3D 25
#define controlButtonsEnd3D (controlButtonsStart3D + maxControlButtons3D)

#define graphStart3D  25  /* the index of g1 */
#define graphSelectStart3D (graphStart3D+maxGraphs)

/* these should be maxControlButtons3D+1.. (be sure to modify view3d.spad) */
#define diagOnOff            (maxControlButtons3D+1)
#define perspectiveOnOff     (maxControlButtons3D+2)
#define clipRegionOnOff      66
#define clipSurfaceOnOff     67

#define query       11


/* misc */

#define spadPressedAButton 100  /* used for communications with the .AXIOM file */
#define colorDef           101
#define moveViewport       102
#define resizeViewport     103
#define changeTitle        104
#define showing2D          105
#define putGraph           106 /* for 2D */
#define getGraph           107 /* for 2D */
#define lightDef           108 /* for 3D */
#define translucenceDef    109 /* for 3D */
#define writeView          110 /* for both */
#define eyeDistanceData    111 /* for 3D */
#define axesColor2D        112 /* for 2D */
```

```
#define unitsColor2D       113 /* for 2D */
#define modifyPOINT        114 /* for 3D */
#define hitherPlaneData    116 /* for 3D */
```

———-

4.2 include/rgb.h

— **include/rgb.h** —

```
/* from bookvol7 chunk include/rgb.h */
typedef struct _RGB {
  float r,g,b;
} RGB ;

typedef struct _HSV {
  float h,s,v;
} HSV ;

typedef struct _HLS {
  float h,l,s;
} HLS ;
```

———-

4.3 include/spadcolors.h

— **include/spadcolors.h** —

```
/* from bookvol7 chunk include/spadcolors.h */
#define numOfColors 240
#define totalHuesConst   27
#define totalShadesConst 5
#define hueEnd  360
#define hueStep 12    /* hueEnd/totalHuesConst */

#define numPlanes 1
#define numColors 10
#define startColor 0
#define endColor   startColor+numColors

#define colorStep (maxColors+1)/numColors
```

```
#define yes 1
#define no 0

#define smoothConst  50
#define saymem(a,b,c) saymemWithLine(a,b,c,0)
#define Colorcells 256   /* KF number of elements in permutation vector */
#define shade 5
#define saturation   0.8

extern int       smoothHue;
extern Colormap  colorMap;
extern int       num;

#define maxColors DisplayCells(dsply,scrn)-1

\getchunk{include/rgb.h}
```

———-

4.4 include/addfile.h1

— **include/addfile.h1** —

```
/* from bookvol7 chunk include/addfile.h1 */
extern FILE * db_file_open(char * db_file);
extern void extend_ht(char * name);
extern FILE * ht_file_open(char * fname , char * aname , char * name);
extern FILE * temp_file_open(char * temp_db_file);
#ifdef _ADDFILE_C
static int build_ht_filename(char * fname , char * aname , char * name);
static int pathname(char * name);
static int strpostfix(char * s , char * t);
#endif
```

———-

4.5 include/all-hyper-proto.h1

— **include/all-hyper-proto.h1** —

```
/* from bookvol7 chunk include/all-hyper-proto.h1 */
\getchunk{include/addfile.h1}
\getchunk{include/readbitmap.h1}
```

```
\getchunk{include/dialog.h1}
\getchunk{include/cond.h1}
\getchunk{include/display.h1}
\getchunk{include/event.h1}
\getchunk{include/ex2ht.h1}
\getchunk{include/form-ext.h1}
\getchunk{include/extent1.h1}
\getchunk{include/extent2.h1}
\getchunk{include/halloc.h1}
\getchunk{include/group.h1}
\getchunk{include/hterror.h1}
\getchunk{include/htinp.h1}
\getchunk{include/hyper.h1}
\getchunk{include/initx.h1}
\getchunk{include/input.h1}
\getchunk{include/keyin.h1}
\getchunk{include/item.h1}
\getchunk{include/lex.h1}
\getchunk{include/parse.h1}
\getchunk{include/macro.h1}
\getchunk{include/parse-paste.h1}
\getchunk{include/parse-aux.h1}
\getchunk{include/parse-input.h1}
\getchunk{include/show-types.h1}
\getchunk{include/parse-types.h1}
\getchunk{include/scrollbar.h1}
\getchunk{include/titlebar.h1}
\getchunk{include/spadint.h1}
\getchunk{include/hash.h1}
\getchunk{include/mem.h1}
```

———-

4.6 include/bsdsignal.h

— **include/bsdsignal.h** —

```
/* from bookvol7 chunk include/bsdsignal.h */
#ifndef _BSDSIGNAL_H_
#define _BSDSIGNAL_H_

#define RestartSystemCalls 1
#define DontRestartSystemCalls 0

typedef void (* SignalHandlerFunc)(int);

#endif     /* _BSDSIGNAL */
```

———-

4.7 include/bsdsignal.h1

— include/bsdsignal.h1 —

```
/* from bookvol7 chunk include/bsdsignal.h1 */
extern SignalHandlerFunc bsdSignal(int  , SignalHandlerFunc  , int );
```

———-

4.8 include/com.h

— include/com.h —

```
/* from bookvol7 chunk include/com.h */
#ifndef _COM_H_
#define _COM_H_

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#if defined(RIOSplatform)
#include <sys/select.h>
#endif

typedef struct {
  int socket;           /* socket number returned by "socket" call */
  int type;             /* socket type (AF_UNIX or AF_INET) */
  int purpose;          /* can be SessionManager, GraphicsServer, etc. */
  int pid;              /* process ID of connected socket */
  int frame;            /* spad interpreter frame (for interpreter windows) */
  int remote_fd;        /* file descriptor of remote socket */
  union {
    struct sockaddr u_addr;
    struct sockaddr_in i_addr;
  } addr;
  char *host_name;      /* name of foreign host if type == AF_INET */
} Sock;

#define MaxClients      150

/* possible socket types (purpose) */
```

```
#define SessionManager  1
#define ViewportServer  2
#define MenuServer      3
#define SessionIO       4
#define BaloonServer    5
#define InterpWindow    6
#define KillSpad        7
#define DebugWindow     8
#define Forker          9
#define AV              10 /*Simon's algebraic viewer */

#define Acknowledge     255

/* Timeout value for connection to remote socket */

#define Forever 0

/* Socket name for local AXIOM server and session manager */

#define SpadServer              "/tmp/.d"
#define SessionServer           "/tmp/.s"
#define SessionIOName           "/tmp/.i"
#define MenuServerName          "/tmp/.h"
#define ForkServerName          "/tmp/.f"


#define MASK_SIZE       (NBBY*sizeof(fd_set))


/* table of dedicated socket types */

extern Sock *purpose_table[];
extern Sock server[];
extern Sock clients[];
extern fd_set socket_mask;
extern fd_set server_mask;

/* Commands sent over the AXIOM session manager or menu socket */

#define CreateFrame             1
#define SwitchFrames            2
#define EndOfOutput             3
#define CallInterp              4
#define EndSession              5
#define LispCommand             6
#define SpadCommand             7
#define SendXEventToHyperTeX    8
#define QuietSpadCommand        9
#define CloseClient             10
#define QueryClients            11
```

```
#define QuerySpad               12
#define NonSmanSession          13
#define KillLispSystem          14

#define CreateFrameAnswer       50

/* Commands from AXIOM menu server to interpreter windows */

#define ReceiveInputLine        100
#define TestLine                101

#endif
```

———-

4.9 include/cond.h1

— **include/cond.h1** —

```
/* from bookvol7 chunk include/cond.h1 */
extern void change_cond(char * label , char * newcond);
extern int check_condition(TextNode * node);
extern void insert_cond(char * label , char * cond);
#ifdef _COND_C
static int check_memostack(TextNode * node);
#endif
```

———-

4.10 include/cursor.h1

— **include/cursor.h1** —

```
/* from bookvol7 chunk include/cursor.h1 */
extern int Cursor_shape(int );
```

———-

4.11 include/debug.h

— **include/debug.h** —

```
/* from bookvol7 chunk include/debug.h */
/* redefine free */
/* #define free hfree*/
```

———-

4.12 include/dialog.h1

— **include/dialog.h1** —

```
/* from bookvol7 chunk include/dialog.h1 */
extern void add_buffer_to_sym(char * buffer , InputItem * sym);
extern void dialog(XEvent * event , KeySym keysym , char * buffer);
extern void draw_inputsymbol(InputItem * sym);
extern void update_inputsymbol(InputItem * sym);
#ifdef _DIALOG_C
static void back_over_char(InputItem * sym);
static void back_over_eoln(InputItem * sym);
static void clear_cursor(InputItem * sym);
static void clear_cursorline(InputItem * sym);
static void dec_line_numbers(LineStruct * line);
static void decrease_line_numbers(LineStruct * line , int am);
static void delete_char(InputItem * sym);
static void delete_eoln(InputItem * sym);
static int delete_one_char(InputItem * sym);
static void delete_rest_of_line(InputItem * sym);
static void draw_cursor(InputItem * sym);
static void enter_new_line(InputItem * sym);
static void inc_line_numbers(LineStruct * line);
static void insert_buffer(char * buffer , InputItem * sym);
static int move_back_one_char(InputItem * sym);
static void move_cursor_backward(InputItem * sym);
static void move_cursor_down(InputItem * sym);
static void move_cursor_end(InputItem * sym);
static void move_cursor_forward(InputItem * sym);
static void move_cursor_home(InputItem * sym);
static void move_cursor_up(InputItem * sym);
static char move_rest_back(LineStruct * line , int size);
static int move_sym_forward(LineStruct * line, int num, int size,
                            InputItem * sym);
static char * mystrncpy(char * buff1 , char * buff2 , int n);
static void overwrite_buffer(char * buffer , InputItem * item);
static void redraw_win(void);
static void tough_enter(InputItem * sym);
#endif
```

———-

4.13 include/display.h1

— **include/display.h1** —

```
/* from bookvol7 chunk include/display.h1 */
extern void expose_page(HyperDocPage * page);
extern void paste_page(TextNode * node);
extern void scroll_page(HyperDocPage * page);
extern void show_page(HyperDocPage * page);
```

———-

4.14 include/edible.h

— **include/edible.h** —

```
/* from bookvol7 chunk include/edible.h */
extern int contNum;
extern struct termios childbuf;   /** the childs normal operating termio   ***/

/***   the terminals mapping of the function keys  ***/
extern unsigned char  _INTR, _QUIT, _ERASE, _KILL, _EOF, _EOL, _RES1, _RES2;
extern short  INS_MODE ;     /** Flag for insert mode      **/
extern short ECHOIT;      /** Flag for echoing **/
extern short PTY;   /* A flag which lets me know whether or not I am
       talking to a socket or a pty. If I am not
       talking to a PTY then I have to do things like echo
      back newlines, and send interuppts with an eoln
      */
/*************************************************************************
    Here are the key mapping my routines need
*************************************************************************/

#define       _ESC   0X1B    /**  A character sent before every arrow key ***/
#define       _LBRACK  0X5B    /**  [       **/
#define       _EOLN    '\n'    /** eoln       **/
#define       _CR      0X0D    /**  cr       **/
#define       _BLANK   0X20    /**  blank       **/
#define       _BKSPC   0X08    /**  backspace       **/
#define       _DEL     0X7F    /**  delete       **/
#define       _BELL    0X07    /***  ring the bell       **/
#define       _INT     0X7F    /***  interrupt       **/
#define       _SQUASH  0X03    /**   kill my process       **/
#define       _CNTRL_W 0X17    /**   cntrl-w, to back up a word       **/
#define       _CARROT  0X5E    /** circumflex       **/
```

```
#define       _TAB     0X09    /** tab forward       **/

#ifndef WCT
#define       _A       0X41    /**  A       **/
#define       _B       0X42    /**  B       **/
#define       _C       0X43    /**  C       **/
#define       _D       0X44    /**  D       **/
#define       _Z       0X5A    /**  Z       **/
#define       _H       0X48    /**  H       **/
#define       _M       0X4D    /**  M       **/
#define       _x       0X78    /**  x       **/
#define       _z       0X7A    /*** z       **/
#define       _twiddle 0X7E    /*** ~       **/
#define       _P       0X50    /*** P       **/
#define       _1       0X31    /*** 1       **/
#define       _2       0X32    /*** 2       **/
#define       _3       0X33    /*** 3       **/
#define       _4       0X34    /*** 4       **/
#define       _5       0X35    /*** 5       **/
#define       _6       0X36    /*** 6       **/
#define       _7       0X37    /*** 7       **/
#define       _8       0X38    /*** 8       **/
#define       _9       0X39    /*** 9       **/
#define       _0       0X30    /*** 0       **/
#define       _q       0X71    /*** q       **/
#endif

#define MAXLINE    1024   /** maximum chars. on a line   ***/
#define MAXBUFF     64  /** maximum lines saved in the buffer
       queue ***/

/***  Here are the constants for my three different modes. ******/
#define CLEFRAW      0
#define CLEFCANONICAL      1
#define CLEFCBREAK      2

extern int mode;   /** One of the above # defines *****/

/**   Here is the structure for storing bound  pf-keys    ***/
typedef struct Fkey
{
  char *str;
  short type;
} fkey;

extern fkey function_key[13] ;   /** strings which replace function
  keys when a key is hit   ***/


extern char editorfilename[];
```

```
/****  Here are a bunch of constant, variable and function defs for edin.c */
#define UP      0  /** Tells the replace buffer command ***/
#define DOWN      1  /**   to look up or down  **/

#define inc(x) ((x+1)%MAXBUFF)  /** returns the increment of the presented
       pointer ***/
#define dec(x) ( ((x-1) < 0) ?(MAXBUFF - 1):(x-1))/** ibid for decrementing */

#define flip(x)   (x?(x=0):(x=1))      /*** flip the bit   ***/

/*
    All the previous commands will now be stored in a double linked list.
  This way when I type a command I just have to circle through this list
*/
typedef struct que_struct {
   char buff[1024];
   int flags[1024];
   struct que_struct *prev, *next;
   } QueStruct;

typedef struct wct {
    char *fname;
    off_t fsize;
    time_t ftime;
    char *fimage;
    int wordc;
    char **wordv;

    struct wct *next;
} Wct;

typedef struct wix {
    Wct *pwct;
    int word;
} Wix;


extern QueStruct *ring;
extern QueStruct *current;
extern int ring_size;
extern int prev_check;
extern int MAXRING;

extern char buff[MAXLINE];    /** Buffers for collecting input and  **/
extern int  buff_flag[MAXLINE];    /**    flags for whether buff chars
   are printing
   or non-printing   **/

extern char in_buff[1024];   /**     buffer for characters read until they are
```

```
      processed   **/
extern int num_read;
extern int num_proc;      /**   num chars processed after a read   **/
extern int buff_pntr;      /**   present length of  buff   **/
extern int curr_pntr;      /** the current position in buff   **/

/** Here are a bunch of macros for edin.c. They are mostly just charcter
comparison stuff   ***/
#define back_word(x) (((*(x) == _5) && (*(x+1) == _9) && \
(*(x+2) == _q))?(1):(0))

#define fore_word(x) (((*(x) == _6) && (*(x+1) == _8) && \
(*(x+2) == _q))?(1):(0))

#define alt_f1(x) (((*(x) == _3) && (*(x+1) == _7) && \
(*(x+2) == _q))?(1):(0))

#define cntrl_end(x) (((*(x) == _4) && (*(x+1) == _8) && \
(*(x+2) == _q))?(1):(0))

#define insert_toggle(x) (((*(x) == _3) && (*(x+1) == _9) && \
(*(x+2) == _q))?(1):(0))

#define end_key(x) (((*(x) == _4) && (*(x+1) == _6) && \
(*(x+2) == _q))?(1):(0))

#define control_char(x) \
     (((x >= 0x01) && (x <= 0x1a))?(1):(0))



/***
   Some global defs needed for emulating a pty. This was taken from guru.h
***/



/* Return an integer that is represented by a character string */
#define ciret(x) ((cintu.c4[0]=(x)[0]), (cintu.c4[1]=(x)[1]), \
  (cintu.c4[2]=(x)[2]), (cintu.c4[3]=(x)[3]), cintu.i4)

/* move an integer (x) to a character string (y) */

#define icmove(x, y) ((cintu.i4=(x)), ((y)[0]=cintu.c4[0]), \
      ((y)[1]=cintu.c4[1]), ((y)[2]=cintu.c4[2]), \
 ((y)[3]=cintu.c4[3]))

/* Return an integer that may not be on an integer boundary */
#define iiret(x) ciret(((char *)&(x)))
```

```
 /* Min of two expressions */
#define min(x, y) ((x)<(y)?(x):(y))

 /* Max of two expressions */
#define max(x, y) ((x)>(y)?(x):(y))
```

———-

4.15 include/edible.h1

— **include/edible.h1** —

```
/* from bookvol7 chunk include/edible.h1 */
extern void check_flip(void);
extern int main(int  , char * []);
extern void catch_signals(void);
extern void init_parent(void);
extern void set_function_chars(void);
extern void hangup_handler(int );
extern void terminate_handler(int );
extern void interrupt_handler(int );
extern void child_handler(int );
extern void alarm_handler(int );
extern void flip_canonical(int );
extern void flip_raw(int );
extern void etc_get_next_line(char *  , int *  , int );
```

———-

4.16 include/edin.h1

— **include/edin.h1** —

```
/* from bookvol7 chunk include/edin.h1 */
extern void init_reader(void);
extern void init_flag(int *  , int );
extern void do_reading(void);
extern void send_line_to_child(void);
extern void insert_buff_nonprinting(int );
extern void prev_buff(void);
extern void next_buff(void);
extern void insert_buff_printing(int );
extern void insert_queue(void);
extern int convert_buffer(char *  , char *  , int *  , int );
```

```
extern void init_buff(char *  , int );
extern void forwardcopy(char *  , char *  , int );
extern void forwardflag_cpy(int *  ,int *  , int );
extern void flagcpy(int *  , int * );
extern void flagncpy(int *  , int *  , int );
extern void send_function_to_child(void);
extern void send_buff_to_child(int );
```

———-

4.17 include/event.h1

— **include/event.h1** —

```
/* from bookvol7 chunk include/event.h1 */
extern void exitHyperDoc(void );
extern void helpForHyperDoc(void );
extern void mainEventLoop(void );
extern void make_window_link(char * name);
extern void quitHyperDoc(void );
extern void get_new_window(void );
#ifdef _EVENT_C
static void set_cursor(HDWindow * window , Cursor state);
static void change_cursor(Cursor state , HDWindow * window);
static void create_window(void );
static void downlink(void );
static HyperDocPage * find_page(TextNode * node);
static void handle_button(int button , XButtonEvent * event);
static void handle_event(XEvent * event);
static void handle_motion_event(XMotionEvent * event);
static int HyperDocErrorHandler(Display * display , XErrorEvent * xe);
static void init_cursor_states(void );
static void killAxiomPage(HyperDocPage * page);
static void make_busy_cursor(HDWindow * window);
static void make_busy_cursors(void );
static void memolink(void );
static void set_error_handlers(void );
static int set_window(Window window);
static void clear_exposures(Window w);
static void kill_page(HyperDocPage * page);
static HyperDocPage * returnlink(void );
static HyperDocPage * uplink(void );
static void windowlink_handler(TextNode * node);
static void lispwindowlink_handler(HyperLink * link);
static HyperDocPage * paste_button(PasteNode * paste);
static HyperLink * findButtonInList(HDWindow * window , int x , int y);
static HyperLink * get_hyper_link(XButtonEvent * event);
```

```
static void init_cursor_state(HDWindow * window);
#endif
```

———-

4.18 include/ex2ht.h1

— **include/ex2ht.h1** —

```
/* from bookvol7 chunk include/ex2ht.h1 */
extern int main(int argc , char * * argv);
extern void openCoverPage(void);
extern void exToHt(char * filename);
extern void closeCoverPage(void);
extern void addFile(char * filename);
extern void closeCoverFile(void);
extern char * allocString(char * s);
extern char * strPrefix(char * prefix , char * s);
extern char * getExTitle(FILE * inFile , char * line);
extern void emitCoverLink(char * name , char * title);
extern void emitHeader(FILE * outFile , char * pageName , char * pageTitle);
extern void emitMenuEntry(char * line , FILE * outFile);
extern void emitSpadCommand(char * line , char * prefix , FILE * outFile);
extern void emitFooter(FILE * outFile);
```

———-

4.19 include/extent1.h1

— **include/extent1.h1** —

```
/* from bookvol7 chunk include/extent1.h1 */
extern void compute_header_extent(HyperDocPage * page);
extern void compute_footer_extent(HyperDocPage * page);
extern void compute_scrolling_extent(HyperDocPage * page);
extern void compute_title_extent(HyperDocPage * page);
extern void compute_text_extent(TextNode * node);
#ifdef _EXTENT1_C
static void compute_begin_items_extent(TextNode * node);
static void compute_bf_extent(TextNode * node);
static void compute_box_extent(TextNode * node);
static void compute_button_extent(TextNode * node);
static void compute_center_extent(TextNode * node);
static void compute_dash_extent(TextNode * node);
```

```
static void compute_em_extent(TextNode * node);
static void compute_ifcond_extent(TextNode * node);
static void compute_image_extent(TextNode * node);
static void compute_input_extent(TextNode * node);
static void compute_ir_extent(TextNode * node);
static void compute_it_extent(TextNode * node);
static void compute_item_extent(TextNode * node);
static void compute_mbox_extent(TextNode * node);
static void compute_mitem_extent(TextNode * node);
static void compute_paste_extent(TextNode * node);
static void compute_pastebutton_extent(TextNode * node);
static void compute_punctuation_extent(TextNode * node);
static void compute_rm_extent(TextNode * node);
static void compute_spadcommand_extent(TextNode * node);
static void compute_spadsrc_extent(TextNode * node);
static void compute_spadsrctxt_extent(TextNode * node);
static void compute_table_extent(TextNode * * node);
static void compute_verbatim_extent(TextNode * node);
static void compute_word_extent(TextNode * node);
static void end_spadcommand_extent(TextNode * node);
static void end_spadsrc_extent(TextNode * node);
static void endbutton_extent(TextNode * node);
static void endif_extent(TextNode * node);
static void endpastebutton_extent(TextNode * node);
#endif
```

———-

4.20 include/extent2.h1

— **include/extent2.h1** —

```
/* from bookvol7 chunk include/extent2.h1 */
extern void init_extents(void );
extern void init_text(void );
extern void init_title_extents(HyperDocPage * page);
extern void insert_bitmap_file(TextNode * node);
extern void insert_pixmap_file(TextNode * node);
extern int max_x(TextNode * node , int Ender);
extern int plh(int height);
extern void start_newline(int distance , TextNode * node);
extern int text_height(TextNode * node , int Ender);
extern int text_width(TextNode * node , int Ender);
extern int total_width(TextNode * node , int Ender);
extern int trailing_space(TextNode * node);
#ifdef _EXTENT2_C
static void center_nodes(TextNode * begin_node , TextNode * end_node);
```

```
static int input_string_width(TextNode * node);
static int punctuation_width(TextNode * node);
static int text_height1(TextNode * node , int Ender);
static int verbatim_width(TextNode * node);
static int width_of_dash(TextNode * node);
static int word_width(TextNode * node);
static int x_value(TextNode * node);
#endif
```

———-

4.21 include/fnct-key.h1

— **include/fnct-key.h1** —

```
/* from bookvol7 chunk include/fnct-key.h1 */
extern void set_editor_key(void);
extern void define_function_keys(void);
extern int get_key(int  , char * );
extern int get_str(int  , char * );
extern void null_fnct(int );
extern void handle_function_key(int  , int );
```

———-

4.22 include/form-ext.h1

— **include/form-ext.h1** —

```
/* from bookvol7 chunk include/form-ext.h1 */
extern void compute_form_page(HyperDocPage * page);
extern int window_width(int cols);
#ifdef _FORM_EXT_C
static int window_height(HyperDocPage * page);
static void form_header_extent(HyperDocPage * page);
static void form_footer_extent(HyperDocPage * page);
static void form_scrolling_extent(HyperDocPage * page);
#endif
```

———-

4.23 include/group.h1

— **include/group.h1** —

```
/* from bookvol7 chunk include/group.h1 */
extern void bf_top_group(void );
extern GroupItem * copy_group_stack(void );
extern void em_top_group(void );
extern void free_group_stack(GroupItem * g);
extern void init_group_stack(void );
extern void init_top_group(void );
extern void line_top_group(void );
extern int pop_group_stack(void );
extern void push_active_group(void );
extern void push_group_stack(void );
extern void push_spad_group(void );
extern void rm_top_group(void );
extern void tt_top_group(void );
extern void center_top_group(void );
```

————-

4.24 include/halloc.h1

— **include/halloc.h1** —

```
/* from bookvol7 chunk include/halloc.h1 */
extern char * halloc(int bytes , char * msg);
```

————-

4.25 include/hash.h

— **include/hash.h** —

```
/* from bookvol7 chunk include/hash.h */
#ifndef _HASH_H_
#define _HASH_H_ 1

typedef struct hash_entry {
  char *key; /* pointer to key data */
  char *data; /* Pointer to entry */
```

```
  struct hash_entry *next; /* Link to next entry */
} HashEntry;

typedef int (*EqualFunction)(void *,void *);
typedef int (*HashcodeFunction)(void *,int);
typedef void (*MappableFunction) (void *);
typedef void (*FreeFunction) (void *);
typedef struct {
  HashEntry **table; /* the actual table */
  int size; /* size of table */
  int num_entries; /* number of elements in a hash table */
  EqualFunction equal; /* equality predicate for keys */
  HashcodeFunction hash_code; /* create hash code for a key */
} HashTable;

#endif
```

———-

4.26 include/hash.h1

— **include/hash.h1** —

```
/* from bookvol7 chunk include/hash.h1 */
extern char * alloc_string(char * str);
extern HashEntry * hash_copy_entry(HashEntry * e);
extern HashTable * hash_copy_table(HashTable * table);
extern void hash_delete(HashTable * table , char * key);
extern char * hash_find(HashTable * table , char * key);
extern void hash_init(HashTable * table, int size, EqualFunction equal,
                      HashcodeFunction hash_code);
extern void free_hash(HashTable * table , FreeFunction free_fun);
extern void hash_insert(HashTable * table , char * data , char * key);
extern void hash_map(HashTable * table , MappableFunction func);
extern char * hash_replace(HashTable * table , char * data , char * key);
extern int string_equal(char * s1 , char * s2);
extern int string_hash(char * s , int size);
```

———-

4.27 include/htadd.h1

— **include/htadd.h1** —

```
/* from bookvol7 chunk include/htadd.h1 */
extern int main(int argc , char * * argv);
#ifdef _HTADD_C
static void add_file(char * dbname , char * name , int fresh);
static void add_new_pages(FILE * temp_db, FILE * new_file,
                          char * addname, char * fullname);
static int build_db_filename(short flag , char * db_dir , char * dbfilename);
static void copy_file(char * f1 , char * f2);
static void delete_db(FILE * db , FILE * temp_db , char * name);
static int delete_file(char * dbname , char * name);
static void get_filename(void);
static void parse_args(char * * argv, char * db_dir,
                       char * * filenames, short * fl);
static void update_db(FILE * db, FILE * temp_db, FILE * new_file,
                      char * addname, char * fullname, int fresh);
static int writable(struct stat buff);
#endif
```

———-

4.28 include/hterror.h1

— **include/hterror.h1** —

```
/* from bookvol7 chunk include/hterror.h1 */
extern void print_page_and_filename(void );
extern void jump(void );
extern void print_token(void );
extern void token_name(int type);
extern void print_next_ten_tokens(void );
extern void htperror(char * msg , int errno);
```

———-

4.29 include/hthits.h1

— **include/hthits.h1** —

```
/* from bookvol7 chunk include/hthits.h1 */
extern void regerr(int code);
extern int main(int argc , char * * argv);
extern void cmdline(int argc , char * * argv);
extern void handleHtdb(void);
extern void badDB(void);
```

```
extern void handleFile(FILE * htdbFile);
extern void handleFilePages(char * fname , int pgc , PgInfo * pgv);
extern void handlePage(FILE * infile , PgInfo * pg);
extern void splitpage(char * buf , char * * ptitle , char * * pbody);
extern void untexbuf(register char * s);
extern void searchPage(char * pgname , char * pgtitle , char * pgbody);
extern void squirt(char * s , int n);
```

———-

4.30 include/htinp.h1

— **include/htinp.h1** —

```
/* from bookvol7 chunk include/htinp.h1 */
extern void ht2_input(void );
extern void make_record(void );
extern void verify_record(void );
extern char * strCopy(char * s);
extern void print_paste_line(FILE * pfile , char * str);
extern void get_spad_output(FILE * pfile , char * command , int com_type);
extern void get_graph_output(char * command , char * pagename , int com_type);
#ifdef _HTINP_C
static void make_input_file_list(void );
static char * make_input_file_name(char * buf , char * filename);
static char * make_paste_file_name(char * buf , char * filename);
static void make_the_input_file(UnloadedPage * page);
static void make_input_file_from_page(HyperDocPage * page);
static int inListAndNewer(char * inputFile , char * htFile);
static void print_paste(FILE * pfile, char * realcom, char * command,
                        char * pagename, int com_type);
static void print_graph_paste(FILE * pfile, char * realcom, char * command,
                              char * pagename, int com_type);
static void send_command(char * command , int com_type);
#endif
```

———-

4.31 include/hyper.h1

— **include/hyper.h1** —

```
/* from bookvol7 chunk include/hyper.h1 */
extern void sigusr2_handler(int sig);
```

```
extern void sigcld_handler(int sig);
extern void clean_socket(void);
extern int main(int argc , char * * argv);
extern void init_page_structs(HDWindow * w);
#ifdef _HYPER_C
static void init_hash(void);
static void make_server_connections(void);
static void check_arguments(void);
#endif
```

———-

4.32 include/initx.h1

— **include/initx.h1** —

```
/* from bookvol7 chunk include/initx.h1 */
extern void change_text(int color , XFontStruct * font);
extern int init_form_window(char * name , int cols);
extern int init_top_window(char * name);
extern void initializeWindowSystem(void);
extern int is_it_850(XFontStruct * fontarg);
#ifdef _INITX_C
static void get_GCs(HDWindow * window);
static int get_border_properties(void);
static int get_color(char * name , char * class , int def , Colormap * map);
static void ingItColors_and_fonts(void);
static void load_font(XFontStruct * * font_info , char * fontname);
static void mergeDatabases(void);
static void open_form_window(void);
static void open_window(Window w);
static void set_name_and_icon(void);
static void set_size_hints(Window w);
#endif
```

———-

4.33 include/input.h1

— **include/input.h1** —

```
/* from bookvol7 chunk include/input.h1 */
extern InputItem * return_item(char * name);
extern void fill_box(Window w , ImageStruct * image);
```

```
extern void toggle_input_box(HyperLink * link);
extern void toggle_radio_box(HyperLink * link);
extern void change_input_focus(HyperLink * link);
extern void next_input_focus(void);
extern void prev_input_focus(void);
extern int delete_item(char * name);
#ifdef _INPUT_C
static void clear_rbs(InputBox * list);
#endif
```

———-

4.34 include/item.h1

— **include/item.h1** —

```
/* from bookvol7 chunk include/item.h1 */
extern void push_item_stack(void);
extern void clear_item_stack(void);
extern void pop_item_stack(void);
extern ItemStack * copy_item_stack(void);
extern void free_item_stack(ItemStack * is);
```

———-

4.35 include/keyin.h1

— **include/keyin.h1** —

```
/* from bookvol7 chunk include/keyin.h1 */
extern void handle_key(XEvent * event);
extern void init_keyin(void);
```

———-

4.36 include/lex.h1

— **include/lex.h1** —

```
/* from bookvol7 chunk include/lex.h1 */
extern int connect_spad(void);
```

```
extern void get_expected_token(int type);
extern void parser_init(void);
extern void init_scanner(void);
extern void save_scanner_state(void);
extern void restore_scanner_state(void);
extern void unget_char(int c);
extern int get_char(void);
extern void unget_token(void);
extern int get_token(void);
extern void push_be_stack(int type , char * id);
extern void check_and_pop_be_stack(int type , char * id);
extern int clear_be_stack(void);
extern int be_type(char * which);
extern int begin_type(void);
extern int end_type(void);
extern void reset_connection(void);
extern int spad_busy(void);
#ifdef _LEX_C
static int get_char1(void );
static void spad_error_handler(void );
static int keyword_type(void );
#endif
```

———-

4.37 include/macro.h1

— **include/macro.h1** —

```
/* from bookvol7 chunk include/macro.h1 */
extern void scan_HyperDoc(void);
extern int number(char * str);
extern ParameterList init_parameter_elem(int number);
extern int push_parameters(ParameterList new);
extern int pop_parameters(void);
extern int parse_macro(void);
extern void parse_parameters(void);
#ifdef _MACRO_C
static char * load_macro(MacroStore * macro);
static void get_parameter_strings(int number , char * macro_name);
#endif
```

———-

4.38 include/mem.h1

— **include/mem.h1** —

```
/* from bookvol7 chunk include/mem.h1 */
extern ButtonList * alloc_button_list(void);
extern CondNode * alloc_condnode(void);
extern HDWindow * alloc_hd_window(void);
extern IfNode * alloc_ifnode(void);
extern InputBox * alloc_inputbox(void);
extern LineStruct * alloc_inputline(int size);
extern TextNode * alloc_node(void);
extern HyperDocPage * alloc_page(char * name);
extern PasteNode * alloc_paste_node(char * name);
extern RadioBoxes * alloc_rbs(void);
extern void free_button_list(ButtonList * bl);
extern void free_hd_window(HDWindow * w);
extern void free_input_item(InputItem * sym , short des);
extern void free_input_list(InputItem * il);
extern void free_node(TextNode * node , short des);
extern void free_page(HyperDocPage * page);
extern void free_patch(PatchStore * p);
extern void free_string(char * str);
extern char * resizeBuffer(int size , char * oldBuf , int * oldSize);
extern PatchStore * alloc_patchstore(void);
#ifdef _MEM_C
static void free_cond(CondNode * cond);
static void free_depend(SpadcomDepend * sd);
static void free_lines(LineStruct * lines);
static void dont_free(void * link);
static void free_if_non_NULL(void * p);
static void free_input_box(InputBox * box);
static void free_paste(PasteNode * paste , short des);
static void free_pastearea(TextNode * node , short des);
static void free_pastebutton(TextNode * node , short des);
static void free_radio_boxes(RadioBoxes * radio);
#endif
```

———-

4.39 include/parse-aux.h1

— **include/parse-aux.h1** —

```
/* from bookvol7 chunk include/parse-aux.h1 */
```

```
extern void add_dependencies(void );
extern FILE * find_fp(FilePosition fp);
extern char * get_input_string(void );
extern HyperLink * make_link_window(TextNode * link_node , int type , int isSubWin);
extern HyperLink * make_paste_window(PasteNode * paste);
extern void make_special_pages(HashTable * pageHashTable);
extern int window_code(Window * w , int size);
extern int window_equal(Window * w1 , Window * w2);
extern char * window_id(Window w);
extern void read_ht_db(HashTable * page_hash , HashTable * macro_hash , HashTable * patch_hash);
extern int get_filename(void);
extern int is_number(char * str);
extern void parser_error(char * str);
extern int get_where(void);
#ifdef _PARSE_AUX_C
static void read_ht_file(HashTable * page_hash , HashTable * macro_hash , HashTable * patch_hash
static HyperDocPage * make_special_page(int type , char * name);
#endif
```

———-

4.40 include/parse.h1

— **include/parse.h1** —

```
/* from bookvol7 chunk include/parse.h1 */
extern void display_page(HyperDocPage * page);
extern void init_parse_patch(HyperDocPage * page);
extern void load_page(HyperDocPage * page);
extern void parse_HyperDoc(void );
extern void parse_from_string(char * str);
extern HyperDocPage * parse_page_from_socket(void );
extern HyperDocPage * parse_page_from_unixfd(void );
#ifdef _PARSE_C
static void end_a_page(void );
static HyperDocPage * format_page(UnloadedPage * ulpage);
static void parse_page(HyperDocPage * page);
static void parse_replacepage(void );
static void start_footer(void );
static void start_scrolling(void );
static void Push_MR(void );
static void Pop_MR(void );
static void parse_title(HyperDocPage * page);
static void parse_header(HyperDocPage * page);
static void init_parse_page(HyperDocPage * page);
#endif
```

———-

4.41 include/parse-input.h1

— **include/parse-input.h1** —

```
/* from bookvol7 chunk include/parse-input.h1 */
extern HyperLink * make_input_window(InputItem * item);
extern HyperLink * make_box_window(InputBox * box , int type);
extern void initialize_default(InputItem * item , char * buff);
extern void parse_inputstring(void);
extern void parse_simplebox(void);
extern void parse_radiobox(void);
extern void init_paste_item(InputItem * item);
extern void repaste_item(void);
extern InputItem * current_item(void);
extern int already_there(char * name);
extern void parse_radioboxes(void);
#ifdef _PARSE_INPUT_C
static void insert_item(InputItem * item);
static void add_box_to_rb_list(char * name , InputBox * box);
static int check_others(InputBox * list);
#endif
```

———-

4.42 include/parse-paste.h1

— **include/parse-paste.h1** —

```
/* from bookvol7 chunk include/parse-paste.h1 */
extern void parse_paste(void);
extern void parse_pastebutton(void);
extern HyperDocPage * parse_patch(PasteNode * paste);
#ifdef _PARSE_PASTE_C
static void load_patch(PatchStore * patch);
#endif
```

———-

4.43 include/parse-types.h1

— **include/parse-types.h1** —

```
/* from bookvol7 chunk include/parse-types.h1 */
extern void parse_begin_items(void );
extern void parse_box(void );
extern void parse_button(void );
extern void parse_centerline(void );
extern void parse_command(void );
extern void parse_env(TextNode * node);
extern void parse_free(void );
extern void parse_help(void );
extern void parse_ifcond(void );
extern void parse_input_pix(void );
extern void parse_item(void );
extern void parse_mbox(void );
extern void parse_mitem(void );
extern void parse_newcond(void );
extern void parse_setcond(void );
extern void parse_spadcommand(TextNode * spad_node);
extern void parse_spadsrc(TextNode * spad_node);
extern void parse_table(void );
extern void parse_value1(void );
extern void parse_value2(void );
extern void parse_verbatim(int type);
#ifdef _PARSE_TYPES_C
static void parse_condnode(void );
static void parse_hasreturnto(void );
#endif
```

———-

4.44 include/pixmap.h1

— **include/pixmap.h1** —

```
/* from bookvol7 chunk include/pixmap.h1 */
extern int file_exists(char * );
extern FILE * zzopen(char *  , char * );
extern void write_pixmap_file(Display *, int, char *, Window, int, int,
                              int, int );
extern int read_pixmap_file(Display *, int, char *, XImage * *, int *, int * );
```

———-

4.45 include/prt.h1

— **include/prt.h1** —

```
/* from bookvol7 chunk include/prt.h1 */
extern void myputchar(char );
extern void clear_buff(void);
extern void move_end(void);
extern void move_home(void);
extern void move_fore_word(void);
extern void move_back_word(void);
extern void delete_current_char(void);
extern void del_print(int  , int );
extern void delete_to_end_of_line(void);
extern void delete_line(void);
extern void printbuff(int  , int );
extern void ins_print(int  , int );
extern void reprint(int );
extern void back_up(int );
extern void back_it_up(int );
extern void print_whole_buff(void);
extern void move_ahead(void);
extern void move_back(void);
extern void back_over_current_char(void);
```

———-

4.46 include/readbitmap.h1

— **include/readbitmap.h1** —

```
/* from bookvol7 chunk include/readbitmap.h1 */
extern XImage * HTReadBitmapFile(Display * display, int screen,
                                 char * filename, int * width, int * height);
extern ImageStruct * insert_image_struct(char * filename);
#ifdef _READBITMAP_C
static int read_hot(FILE * fd , char Line[] , int * x_hot , int * y_hot);
static int read_w_and_h(FILE * fd, unsigned int * width,
                        unsigned int * height);
#endif
```

———-

4.47 include/scrollbar.h1

— **include/scrollbar.h1** —

```
/* from bookvol7 chunk include/scrollbar.h1 */
extern void calculateScrollBarMeasures(void );
extern void drawScrollLines(void );
extern void hideScrollBars(HDWindow * hdWindow);
extern void getScrollBarMinimumSize(int * width , int * height);
extern void linkScrollBars(void );
extern void makeScrollBarWindows(void );
extern void moveScroller(HDWindow * hdWindow);
extern void scrollDown(void );
extern void scrollDownPage(void );
extern void scrollScroller(XButtonEvent * event);
extern void scrollToFirstPage(void );
extern void scrollUp(void );
extern void scrollUpPage(void );
extern void showScrollBars(HDWindow * hdWindow);
#ifdef _SCROLLBAR_C
static int ch(int height);
static void changeWindowBackgroundPixmap(Window window , Pixmap pixmap);
static void drawScroller3DEffects(HDWindow * hdWindow, int x1, int y1,
                                  int x2, int y2);

#endif
```

———-

4.48 include/show-types.h1

— **include/show-types.h1** —

```
/* from bookvol7 chunk include/show-types.h1 */
extern void show_text(TextNode * node , int Ender);
#ifdef _SHOW_TYPES_C
static void show_image(TextNode * node , GC gc);
static void show_input(TextNode * node);
static void show_link(TextNode * node);
static void show_paste(TextNode * node);
static void show_pastebutton(TextNode * node);
static void show_simple_box(TextNode * node);
static void show_spadcommand(TextNode * node);
#endif
```

———-

4.49 include/sockio-c.h1

— **include/sockio-c.h1** —

```
/* from bookvol7 chunk include/sockio-c.h1 */
extern int get_int(Sock * );
extern char * get_string(Sock * );
extern double get_float(Sock * );
extern Sock * connect_to_local_server(char *  , int  , int );
extern int sread(Sock *  , char *  , int  , char * );
extern double plus_infinity(void );
extern double minus_infinity(void );
extern double NANQ(void );
extern void sigpipe_handler(int );
extern int wait_for_client_read(Sock *  , char *  , int  , char * );
extern int wait_for_client_write(Sock *  , char *  , int  , char * );
extern int swrite(Sock *  , char *  , int  , char * );
extern int sselect(int  , fd_set *  , fd_set *  , fd_set *  , void * );
extern int fill_buf(Sock *  , char *  , int  , char * );
extern int sock_get_int(int );
extern int get_ints(Sock *  , int *  , int );
extern int sock_get_ints(int  , int *  , int );
extern int send_int(Sock *  , int );
extern int sock_send_int(int  , int );
extern int send_ints(Sock *  , int *  , int );
extern int sock_send_ints(int  , int *  , int );
extern int send_string(Sock *  , char * );
extern int send_string_len(Sock *  , char *  , int );
extern int sock_send_string(int  , char * );
extern int sock_send_string_len(int  , char *  , int );
extern int send_strings(Sock *  , char * *  , int );
extern int sock_send_strings(int  , char * *  , int );
extern char * sock_get_string(int );
extern char * get_string_buf(Sock *  , char *  , int );
extern char * sock_get_string_buf(int  , char *  , int );
extern int get_strings(Sock *  , char * *  , int );
extern int sock_get_strings(int  , char * *  , int );
extern int send_float(Sock *  , double );
extern int sock_send_float(int  , double );
extern int send_sfloats(Sock *  , float *  , int );
extern int sock_send_sfloats(int  , float *  , int );
extern int send_floats(Sock *  , double *  , int );
extern int sock_send_floats(int  , double *  , int );
extern double sock_get_float(int );
extern int get_sfloats(Sock *  , float *  , int );
extern int sock_get_sfloats(int  , float *  , int );
extern int get_floats(Sock *  , double *  , int );
extern int sock_get_floats(int  , double *  , int );
```

```
extern int wait_for_client_kill(Sock *  , int );
extern int sock_get_remote_fd(int );
extern int send_signal(Sock *  , int );
extern int sock_send_signal(int  , int );
extern int send_wakeup(Sock * );
extern int sock_send_wakeup(int );
extern Sock * connect_to_local_server_new(char *  , int  , int );
extern void remote_stdio(Sock * );
extern void init_purpose_table(void );
extern int make_server_number(void );
extern void close_socket(int  , char * );
extern int make_server_name(char *  , char * );
extern int open_server(char * );
extern int accept_connection(Sock * );
extern void get_socket_type(Sock * );
extern int sock_accept_connection(int );
extern void redirect_stdio(Sock * );
extern void init_socks(void );
extern int server_switch(void );
extern void flush_stdout(void );
extern void print_line(char * );
```

———-

4.50 include/spadbuf.h1

— **include/spadbuf.h1** —

```
/* from bookvol7 chunk include/spadbuf.h1 */
extern int main(int argc , char * * argv);
#ifdef _SPADBUF_C
static void spadbuf_inter_handler(int sig);
static void spadbuf_function_chars(void);
static void interp_io(void);
static void init_parent(void);
#endif
```

———-

4.51 include/spadcolors.h1

— **include/spadcolors.h1** —

```
/* from bookvol7 chunk include/spadcolors.h1 */
```

```
extern RGB HSVtoRGB(HSV );
extern RGB HLStoRGB(HLS );
extern float value(float  , float  , float );
extern int makeColors(Display *  , int  , Colormap *  , unsigned long * *  , int * );
extern int makePermVector(Display *  , int  , unsigned long * * );
extern int makeNewColorMap(Display *  , Colormap  , int );
extern unsigned long XPixelColor(int );
extern void FreePixels(Display *  , Colormap  , int );
extern int AllocCells(Display *  , Colormap  , int );
```

———-

4.52 include/spadint.h1

— **include/spadint.h1** —

```
/* from bookvol7 chunk include/spadint.h1 */
extern HyperDocPage * issue_server_command(HyperLink * link);
extern HyperDocPage * issue_unixlink(TextNode * node);
extern char * print_to_string(TextNode * command);
extern void issue_spadcommand(HyperDocPage * page , TextNode * command ,
                              int immediate , int type);
extern Sock * accept_menu_connection(Sock * server_sock);
extern char * print_to_string1(TextNode * command , int * sizeBuf);
extern int issue_serverpaste(TextNode * command);
extern void issue_unixcommand(TextNode * node);
extern int issue_unixpaste(TextNode * node);
extern void service_session_socket(void);
extern void send_lisp_command(char * command);
extern void escape_string(char * s);
extern void unescape_string(char * s);
extern char * print_source_to_string1(TextNode * command , int * sizeBuf);
extern char * print_source_to_string(TextNode * command);
#ifdef _SPADINT_C
static void start_user_buffer(HyperDocPage * page);
static void clear_execution_marks(HashTable * depend_hash);
static void issue_dependent_commands(HyperDocPage * page, TextNode * command,
                                     int type);
static void send_pile(Sock * sock , char * str);
static void mark_as_executed(HyperDocPage * page, TextNode * command,
                             int type);
static void accept_menu_server_connection(HyperDocPage * page);
static void switch_frames(void );
static void close_client(int pid);
#endif
```

———-

4.53 include/titlebar.h1

— **include/titlebar.h1** —

```
/* from bookvol7 chunk include/titlebar.h1 */
extern void getTitleBarMinimumSize(int * width , int * height);
extern void linkTitleBarWindows(void);
extern void makeTitleBarWindows(void);
extern void showTitleBar(void);
#ifdef _TITLEBAR_C
static void readTitleBarImages(void);
#endif
```

———-

4.54 include/util.h1

— **include/util.h1** —

```
/* from bookvol7 chunk include/util.h1 */
extern int checker(int  , int  , char * );
extern char * getmemWithLine(int  , char *  , int );
extern char * saymemWithLine(char *  , int  , int  , int );
extern void myfree(void *  , int );
extern XPoint getWindowPositionXY(Display *  , Window );
extern XPoint getWindowSizeXY(Display *  , Window );
```

———-

4.55 include/wct.h1

— **include/wct.h1** —

```
/* from bookvol7 chunk include/wct.h1 */
extern time_t ftime(char * );
extern void fatal(char *  , char * );
extern off_t fsize(char * );
extern Wix * scanWct(Wct *  , char * );
extern void reintern1Wct(Wct * );
extern Wix * rescanWct(void);
extern void skimWct(Wct * );
extern void skim1Wct(Wct * );
```

```
extern void printTime(long * );
extern int skimString(char *  , int  , int  , int );
extern int prChar(int );
extern Wct * reread1Wct(Wct * );
extern void sfatal(char * );
extern Wct * read1Wct(char * );
extern Wct * nconcWct(Wct *  , Wct * );
extern void sortWct(Wct * );
extern void sort1Wct(Wct * );
extern int mystrcmp(const void  *  , const void * );
extern void burstWct(Wct * );
extern void burst1Wct(Wct * );
extern Wct * intern1Wct(char * );
extern void load_wct_file(char * );
extern void  skim_wct(void);
extern void  rescan_wct(void);
extern void  find_wct(void);
```

———-

Chapter 5

Shared Code

BeStruct

— **BeStruct** —

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

BeStruct *top_be_stack;
```

————-

5.1 Shared Code for file handling

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;
}
```

———-

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;
}
```

———-

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);
    }
}
```

———-

pathname

— **pathname** —

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

———-

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);
}
```

———-

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);
}
```

———-

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;
}
```

———-

5.2 Shared Code for Hash Table Handling

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;
}
```

———-

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;
}
```

———-

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);
  }
}
```

———-

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++;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
        }
}
```

———-

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);
}
```

———-

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 */
```

———-

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;
}
```

———-

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;
}
```

———-

stringEqual

Test strings for equality.

— **stringEqual** —

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

———-

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);
}
```

———-

5.3 Shared Code for Error Handling

jump

— **jump** —

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

———-

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);
}
```

———-

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);
}
```

———-

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");
}
```

———-

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);
}
```

———-

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;*/
    }
}
```

———-

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();
}
```

———-

5.4 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

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);
    }
}
```

———-

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;
}
```

———-

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;
}
```

———-

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);
}
```

———-

ungetChar

Return the character to the input stream.

— **ungetChar** —

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

———-

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;
}
```

———-

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;
    }
}
```

———-

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);
}
```

———-

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;
}
```

———-

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;
}
```

———-

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();
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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);
    }
}
```

———-

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);
}
```

———-

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();
    }
}
```

———-

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 */
```

———-

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;
}
```

———-

5.5 htadd shared code

— **htadd shared code** —

```
\getchunk{include/bsdsignal.h}
\getchunk{include/bsdsignal.h1}
\getchunk{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"
  };

\getchunk{include/token.h}
\getchunk{spadErrorHandler}
\getchunk{spadBusy}
\getchunk{connectSpad}
\getchunk{resetConnection}
\getchunk{pathname}
\getchunk{BeStruct}
\getchunk{strpostfix}
\getchunk{extendHT}
\getchunk{buildHtFilename}
\getchunk{htFileOpen}
\getchunk{tempFileOpen}
\getchunk{halloc}
\getchunk{hashInit}
\getchunk{hashInsert}
\getchunk{hashDelete}
```

```
\getchunk{hashMap}
\getchunk{hashFind}
\getchunk{hashReplace}
\getchunk{freeHash}
\getchunk{stringHash}
\getchunk{stringEqual}
\getchunk{allocString}
\getchunk{jump}
\getchunk{tokenName}
\getchunk{printToken}
\getchunk{printPageAndFilename}
\getchunk{printNextTenTokens}
\getchunk{parserInit}
\getchunk{initScanner}
\getchunk{saveScannerState}
\getchunk{restoreScannerState}
\getchunk{ungetChar}
\getchunk{getExpectedToken}
\getchunk{ungetToken}
\getchunk{getChar1}
\getchunk{getChar}
\getchunk{getToken}
\getchunk{pushBeStack}
\getchunk{clearBeStack}
\getchunk{checkAndPopBeStack}
\getchunk{beType}
\getchunk{beginType}
\getchunk{endType}
\getchunk{keywordType}
```

———-

5.6 hypertex shared code

— hypertex shared code —

```
\getchunk{include/bsdsignal.h}
\getchunk{include/bsdsignal.h1}
\getchunk{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"
  };

\getchunk{include/token.h}
\getchunk{spadErrorHandler}
\getchunk{spadBusy}
\getchunk{connectSpad}
\getchunk{resetConnection}
\getchunk{pathname}
\getchunk{BeStruct}
\getchunk{strpostfix}
\getchunk{extendHT}
```

```
\getchunk{buildHtFilename}
\getchunk{htFileOpen}
\getchunk{tempFileOpen}
\getchunk{halloc}
\getchunk{hashInit}
\getchunk{hashInsert}
\getchunk{hashDelete}
\getchunk{hashMap}
\getchunk{hashFind}
\getchunk{hashReplace}
\getchunk{freeHash}
\getchunk{stringHash}
\getchunk{stringEqual}
\getchunk{allocString}
\getchunk{jump}
\getchunk{tokenName}
\getchunk{printToken}
\getchunk{printPageAndFilename}
\getchunk{printNextTenTokens}
\getchunk{parserInit}
\getchunk{initScanner}
\getchunk{saveScannerState}
\getchunk{restoreScannerState}
\getchunk{ungetChar}
\getchunk{getExpectedToken}
\getchunk{ungetToken}
\getchunk{getChar1}
\getchunk{getChar}
\getchunk{getToken}
\getchunk{pushBeStack}
\getchunk{clearBeStack}
\getchunk{checkAndPopBeStack}
\getchunk{beType}
\getchunk{beginType}
\getchunk{endType}
\getchunk{keywordType}
```

Chapter 6

Shared include files

6.1 debug.c

— **debug.c** —

```
\getchunk{include/debug.h}

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

————-

6.2 include/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.

— include/hyper.h —

```
/* from bookvol7 chunk include/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>

\getchunk{include/com.h}
\getchunk{include/token.h}
\getchunk{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) || defined(AIX370platform)
#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) || defined (HP9platform)  || d
#define RmFontDefault "-adobe-courier-medium-r-normal--18-*-*-*-m-*-iso8859-1"
#define TtFontDefault "-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-*-iso8859-1"
#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 7

The spadbuf function

7.1 spadbuf Call Graph

This was generated by the GNU cflow program with the argument list. Note that the line:NNNN numbers refer to the line in the code after it has been tangled from this file.

```
cflow --emacs -l -n -b -T --omit-arguments spadbuf.c
```

```
;; This file is generated by GNU cflow 1.3. -*- cflow -*-
    2 {   0} +-main() <int main () line:150>
    3 {   1}   +-fopen()
    4 {   1}   +-fprintf()
    5 {   1}   +-exit()
    6 {   1}   +-load_wct_file()
    7 {   1}   +-skim_wct()
    8 {   1}   +-connect_to_local_server()
    9 {   1}   +-bsdSignal()
   10 {   1}   +-spadbufInterHandler() <void spadbufInterHandler () line:55>
   11 {   2}     \-send_signal()
   12 {   1}   +-send_string()
   13 {   1}   +-initParent() <void initParent () line:116>
   14 {   2}   | +-tcgetattr()
   15 {   2}   | +-perror()
   16 {   2}   | +-exit()
   17 {   2}   | +-tcsetattr()
   18 {   2}   | +-spadbufFunctionChars()
               | | <void spadbufFunctionChars () line:59>
   19 {   2}   | \-Cursor_shape()
   20 {   1}   +-define_function_keys()
   21 {   1}   +-init_reader()
   22 {   1}   \-interpIO() <void interpIO () line:70>
   23 {   2}     +-FD_ZERO()
```

```
24 {   2}     +-FD_SET()
25 {   2}     +-sselect()
26 {   2}     +-perror()
27 {   2}     +-FD_ISSET()
28 {   2}     +-sread()
29 {   2}     +-write()
30 {   2}     +-get_int()
31 {   2}     +-exit()
32 {   2}     +-get_string_buf()
33 {   2}     +-strlen()
34 {   2}     +-clear_buff()
35 {   2}     +-do_reading()
36 {   2}     \-read()
```

7.2 Constants and Headers

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>
```

————-

Local includes

— **spadbuf** —

```
\getchunk{include/debug.h}
\getchunk{include/bsdsignal.h}
\getchunk{include/edible.h}
\getchunk{include/com.h}
\getchunk{include/spadbuf.h1}
\getchunk{include/bsdsignal.h1}
\getchunk{include/sockio-c.h1}
\getchunk{include/edin.h1}
```

```
\getchunk{include/wct.h1}
\getchunk{include/prt.h1}
\getchunk{include/cursor.h1}
\getchunk{include/fnct-key.h1}
```

———-

7.3 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 */
```

———-

7.4 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 */
```

———-

7.5 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

spadbufInterHandler

— **spadbuf** —

```
static void spadbufInterHandler(int sig) {
    send_signal(session_sock, SIGUSR2);
}
```

———-

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;
}
```

———-

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();
        }
    }
}
```

— **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);
}
```

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 8

The ex2ht function

8.1 ex2ht Call Graph

This was generated by the GNU cflow program with the argument list. Note that the line:NNNN numbers refer to the line in the code after it has been tangled from this file.

```
cflow --emacs -l -n -b -T --omit-arguments ex2ht.c
```

```
;; This file is generated by GNU cflow 1.3. -*- cflow -*-
    2 {   0} +-main() <int main () line:180>
    3 {   1}   +-fprintf()
    4 {   1}   +-openCoverPage() <void openCoverPage () line:141>
    5 {   2}   | +-fopen()
    6 {   2}   | +-fprintf()
    7 {   2}   | \-exit()
    8 {   1}   +-exToHt() <void exToHt () line:47>
    9 {   2}   | +-fopen()
   10 {   2}   | +-fprintf()
   11 {   2}   | +-strcpy()
   12 {   2}   | +-strcat()
   13 {   2}   | +-strlen()
   14 {   2}   | +-allocString() <char *allocString () line:20>
   15 {   3}   | | +-malloc()
   16 {   3}   | | +-strlen()
   17 {   3}   | | \-strcpy()
   18 {   2}   | +-getExTitle() <char *getExTitle () line:36>
   19 {   3}   |   +-fgets()
   20 {   3}   |   +-strPrefix() <char *strPrefix () line:26>
   21 {   3}   |   +-strlen()
   22 {   3}   |   \-fprintf()
   23 {   2}   | +-emitCoverLink() <void emitCoverLink () line:161>
   24 {   3}   | | \-fprintf()
```

```
  25 {   2}   | +-emitHeader() <void emitHeader () line:103>
  26 {   3}   | | \-fprintf()
  27 {   2}   | +-fgets()
  28 {   2}   | +-strPrefix() <char *strPrefix () line:26> [see 20]
  29 {   2}   | +-emitMenuEntry() <void emitMenuEntry () line:112>
  30 {   3}   | | \-fprintf()
  31 {   2}   | +-emitSpadCommand() <void emitSpadCommand () line:125>
  32 {   3}   | | \-fprintf()
  33 {   2}   | +-emitFooter() <void emitFooter () line:108>
  34 {   3}   | | \-fprintf()
  35 {   2}   | +-fclose()
  36 {   2}   | +-stat()
  37 {   2}   | \-timercmp()
  38 {   1}   +-closeCoverPage() <void closeCoverPage () line:152>
  39 {   2}     \-fprintf()
  40 {   1}   +-addFile() <void addFile () line:165>
  41 {   2}     +-fopen()
  42 {   2}     +-fprintf()
  43 {   2}     +-exit()
  44 {   2}     +-getc()
  45 {   2}     +-putc()
  46 {   2}     +-fclose()
  47 {   2}     \-unlink()
  48 {   1}   \-closeCoverFile() <void closeCoverFile () line:156>
  49 {   2}     +-fclose()
  50 {   2}     \-utimes()
```

8.2 ex2ht Source Code

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

8.3 Constants and Headers

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>
```

———-

Local includes

— ex2ht —

```
\getchunk{include/debug.h}
\getchunk{include/ex2ht.h1}
```

———-

8.4 defines

— ex2ht —

```
#define MaxLineLength 512
#define MaxFiles      100
```

———-

8.5 local variables

— ex2ht —

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

———-

8.6 Code

allocString

— ex2ht —

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

———-

strPrefix

— **ex2ht** —

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

———-

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;
}
```

———-

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;
    }
}
```

———-

emitHeader

— **ex2ht** —

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

———-

emitFooter

— **ex2ht** —

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

———-

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);
}
```

———-

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);
}
```

———-

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");
}
```

———-

closeCoverPage

— **ex2ht** —

```
void closeCoverPage(void) {
    fprintf(coverFile, "}\\endscroll\\end{page}\n\n");
}
```

———-

closeCoverFile

— **ex2ht** —

```
void closeCoverFile(void) {
    fclose(coverFile);
    utimes("coverex.ht",latest_date);
}
```

———-

emitCoverLink

— **ex2ht** —

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

———-

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);
}
```

———-

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 9

The htadd command

9.1 htadd Call Graph

This was generated by the GNU cflow program with the argument list. Note that the line:NNNN numbers refer to the line in the code after it has been tangled from this file.

```
cflow --emacs -l -n -b -T --omit-arguments htadd.c
```

```
;; This file is generated by GNU cflow 1.3. -*- cflow -*-
    2 {   0} +-main() <int main () line:2528>
    3 {   1}   +-parseArgs() <void parseArgs () line:2188>
    4 {   2}   | +-strcmp()
    5 {   2}   | +-fprintf()
    6 {   2}   | +-exit()
    7 {   2}   | \-strcpy()
    8 {   1}   +-fprintf()
    9 {   1}   +-parserInit() <void parserInit () line:1611>
   10 {   2}   | +-hashInit() <void hashInit () line:1376>
   11 {   3}   |   \-halloc() <char *halloc () line:1355>
   12 {   4}   |     +-fopen()
   13 {   4}   |     +-malloc()
   14 {   4}   |     +-fprintf()
   15 {   4}   |     +-sprintf()
   16 {   4}   |     \-exit()
   17 {   2}   | +-stringEqual() <int stringEqual () line:1470>
   18 {   3}   |   \-strcmp()
   19 {   2}   | +-stringHash() <int stringHash () line:1462>
   20 {   2}   | +-halloc() <char *halloc () line:1355> [see 11]
   21 {   2}   | \-hashInsert() <void hashInsert () line:1389>
   22 {   3}   |   +-halloc() <char *halloc () line:1355> [see 11]
   23 {   3}   |   \-fprintf()
   24 {   1}   +-buildDBFilename() <int buildDBFilename () line:2241>
```

```
25 {   2}     +-getenv()
26 {   2}     +-fprintf()
27 {   2}     +-sprintf()
28 {   2}     +-strcpy()
29 {   2}     +-stat()
30 {   2}     +-perror()
31 {   2}     +-exit()
32 {   2}     \-writable() <int writable () line:2224>
33 {   3}       +-geteuid()
34 {   3}       +-getegid()
35 {   3}       \-fprintf()
36 {   1}   +-unlink()
37 {   1}   +-deleteFile() <int deleteFile () line:2473>
38 {   2}   | +-strcpy()
39 {   2}   | +-extendHT() <void extendHT () line:1223>
40 {   3}   | | +-strpostfix() <int strpostfix () line:1213>
41 {   4}   | | | \-strlen()
42 {   3}   | | \-strcat()
43 {   2}   | +-fopen()
44 {   2}   | +-fprintf()
45 {   2}   | +-tempFileOpen() <FILE *tempFileOpen () line:1343>
46 {   3}   | | +-strcpy()
47 {   3}   | | +-strcat()
48 {   3}   | | +-fopen()
49 {   3}   | | +-perror()
50 {   3}   | | \-exit()
51 {   2}   | +-deleteDB() <void deleteDB () line:2495>
52 {   3}   | | +-initScanner() <void initScanner () line:1628>
53 {   4}   | |   +-getenv()
54 {   4}   | |   \-strcmp()
55 {   3}   | | +-getChar() <int getChar () line:1775>
56 {   4}   | | | \-getChar1() <int getChar1 () line:1718>
57 {   5}   | | |   +-getc()
58 {   5}   | | |   +-get_int()
59 {   5}   | | |   +-spadErrorHandler()
           | | |   |  <void spadErrorHandler () line:1149>
60 {   6}   | | |   | +-longjmp()
61 {   6}   | | |   | +-fprintf()
62 {   6}   | | |   | \-exit()
63 {   5}   | | |   +-get_string_buf()
64 {   5}   | | |   \-fprintf()
65 {   3}   | | +-getFilename() <void getFilename () line:2442>
66 {   4}   | | | +-getChar() <int getChar () line:1775> [see 55]
67 {   4}   | | | +-whitespace()
68 {   4}   | | | +-fprintf()
69 {   4}   | | | +-exit()
70 {   4}   | | | +-filedelim()
71 {   4}   | | | \-ungetChar() <void ungetChar () line:1685>
72 {   3}   | | +-allocString() <char *allocString () line:1474>
73 {   4}   | |   +-halloc() <char *halloc () line:1355> [see 11]
```

```
 74 {   4}   | |   +-strlen()
 75 {   4}   | |   \-strcpy()
 76 {   3}   | | +-getToken() <int getToken () line:1820> (R)
 77 {   4}   | | | +-strcpy()
 78 {   4}   | | | +-free()
 79 {   4}   | | | +-getChar() <int getChar () line:1775> [see 55]
 80 {   4}   | | | +-whitespace()
 81 {   4}   | | | +-ungetChar() <void ungetChar () line:1685> [see 71]
 82 {   4}   | | | +-getToken() <int getToken () line:1820>
             | | | | (recursive: see 76) [see 76]
 83 {   4}   | | | +-isalpha()
 84 {   4}   | | | +-keywordType() <int keywordType () line:2150> (R)
 85 {   5}   | | |   +-hashFind() <char *hashFind () line:1424>
 86 {   5}   | | |   +-beginType() <int beginType () line:2088> (R)
 87 {   6}   | | |     +-beType() <int beType () line:2020> (R)
 88 {   7}   | | |       +-getExpectedToken()
             | | |       | <void getExpectedToken () line:1691> (R)
 89 {   8}   | | |         +-getToken() <int getToken () line:1820>
             | | |         | (recursive: see 76) [see 76]
 90 {   8}   | | |         +-tokenName() <void tokenName () line:1489>
 91 {   9}   | | |         | +-strcpy()
 92 {   9}   | | |         | \-sprintf()
 93 {   8}   | | |         +-fprintf()
 94 {   8}   | | |         +-printPageAndFilename()
             | | |         | <void printPageAndFilename () line:1571>
 95 {   9}   | | |         | +-sprintf()
 96 {   9}   | | |         | \-fprintf()
 97 {   8}   | | |         +-printNextTenTokens()
             | | |         | <void printNextTenTokens () line:1598> (R)
 98 {   9}   | | |         | +-fprintf()
 99 {   9}   | | |         | +-getToken() <int getToken () line:1820>
             | | |         | | (recursive: see 76) [see 76]
100 {   9}   | | |         | \-printToken() <void printToken () line:1561>
101 {  10}   | | |         |   +-printf()
102 {  10}   | | |         |   +-tokenName()
             | | |         |   | <void tokenName () line:1489> [see 90]
103 {  10}   | | |         |   \-fflush()
104 {   8}   | | |         +-longjmp()
105 {   8}   | | |         \-exit()
106 {   7}   | | |       \-strcmp()
107 {   6}   | | |     +-fprintf()
108 {   6}   | | |     +-printPageAndFilename()
             | | |     | <void printPageAndFilename () line:1571> [see 94]
109 {   6}   | | |     +-printNextTenTokens()
             | | |     | <void printNextTenTokens () line:1598> (R) [see 97]
110 {   6}   | | |     +-jump() <void jump () line:1481>
111 {   7}   | | |       +-exit()
112 {   7}   | | |       +-longjmp()
113 {   7}   | | |       \-fprintf()
114 {   6}   | | |     \-pushBeStack() <void pushBeStack () line:1974>
```

```
115 {   7}   | | |       +-halloc() <char *halloc () line:1355> [see 11]
116 {   7}   | | |       \-allocString()
             | | |         <char *allocString () line:1474> [see 72]
117 {   5}   | | |   \-endType() <int endType () line:2113> (R)
118 {   6}   | | |     +-beType() <int beType () line:2020> (R) [see 87]
119 {   6}   | | |     +-fprintf()
120 {   6}   | | |     +-printPageAndFilename()
             | | |     | <void printPageAndFilename () line:1571> [see 94]
121 {   6}   | | |     +-printNextTenTokens()
             | | |     | <void printNextTenTokens () line:1598> (R) [see 97]
122 {   6}   | | |     +-jump() <void jump () line:1481> [see 110]
123 {   6}   | | |     \-checkAndPopBeStack()
             | | |       <void checkAndPopBeStack () line:1996> (R)
124 {   7}   | | |       +-fprintf()
125 {   7}   | | |       +-printPageAndFilename()
             | | |       | <void printPageAndFilename () line:1571> [see 94]
126 {   7}   | | |       +-printNextTenTokens()
             | | |       | <void printNextTenTokens () line:1598> (R)
             | | |       | [see 97]
127 {   7}   | | |       +-jump() <void jump () line:1481> [see 110]
128 {   7}   | | |       \-free()
129 {   4}   | | | +-isdigit()
130 {   4}   | | | \-delim()
131 {   3}   | | +-atoi()
132 {   3}   | | +-strcmp()
133 {   3}   | | +-fprintf()
134 {   3}   | | +-putc()
135 {   3}   | | \-free()
136 {   2}   | +-fclose()
137 {   2}   | +-copyFile() <void copyFile () line:2429>
138 {   3}   | | +-fopen()
139 {   3}   | | +-getc()
140 {   3}   | | +-putc()
141 {   3}   | | \-fclose()
142 {   2}   | \-unlink()
143 {   1}   \-addfile() <void addfile () line:2285>
144 {   2}     +-htFileOpen() <FILE *htFileOpen () line:1326>
145 {   3}     | +-buildHtFilename() <int buildHtFilename () line:1231>
146 {   4}     | | +-cwd()
147 {   4}     | | +-getcwd()
148 {   4}     | | +-strcpy()
149 {   4}     | | +-strcat()
150 {   4}     | | +-strlen()
151 {   4}     | | +-fprintf()
152 {   4}     | | +-exit()
153 {   4}     | | +-extendHT() <void extendHT () line:1223> [see 39]
154 {   4}     | | +-access()
155 {   4}     | | +-pathname() <int pathname () line:1198>
156 {   4}     | | +-getenv()
157 {   4}     | | +-halloc() <char *halloc () line:1355> [see 11]
```

```
158 {   4}     | | \-strcmp()
159 {   3}     | +-fprintf()
160 {   3}     | +-exit()
161 {   3}     | +-fopen()
162 {   3}     | \-perror()
163 {   2}     +-fopen()
164 {   2}     +-fprintf()
165 {   2}     +-exit()
166 {   2}     +-tempFileOpen() <FILE *tempFileOpen () line:1343> [see 45]
167 {   2}     +-updateDB() <void updateDB () line:2327>
168 {   3}     | +-addNewPages() <void addNewPages () line:2378>
169 {   4}     |   +-stat()
170 {   4}     |   +-fprintf()
171 {   4}     |   +-initScanner() <void initScanner () line:1628> [see 52]
172 {   4}     |   +-getToken() <int getToken () line:1820> (R) [see 76]
173 {   4}     |   +-Special()
174 {   4}     |   +-ptype()
175 {   4}     |   +-exit()
176 {   4}     |   \-printf()
177 {   3}     | +-initScanner() <void initScanner () line:1628> [see 52]
178 {   3}     | +-getChar() <int getChar () line:1775> [see 55]
179 {   3}     | +-getFilename() <void getFilename () line:2442> [see 65]
180 {   3}     | +-allocString() <char *allocString () line:1474> [see 72]
181 {   3}     | +-getToken() <int getToken () line:1820> (R) [see 76]
182 {   3}     | +-atoi()
183 {   3}     | +-strcmp()
184 {   3}     | +-saveScannerState() <void saveScannerState () line:1646>
185 {   4}     | | \-halloc() <char *halloc () line:1355> [see 11]
186 {   3}     | +-restoreScannerState()
                |   <void restoreScannerState () line:1662>
187 {   4}     |   +-fprintf()
188 {   4}     |   +-exit()
189 {   4}     |   +-fseek()
190 {   4}     |   \-free()
191 {   3}     | +-fprintf()
192 {   3}     | +-putc()
193 {   3}     | \-free()
194 {   2}     +-fclose()
195 {   2}     +-copyFile() <void copyFile () line:2429> [see 137]
196 {   2}     \-unlink()
```

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
...
```

9.2 Constants and Headers

System includes

— **htadd** —

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

————-

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;
```

————-

Local includes

— **htadd** —

```
\getchunk{include/hyper.h}
```

```
\getchunk{include/htadd.h1}
\getchunk{include/addfile.h1}
\getchunk{include/halloc.h1}
\getchunk{include/hash.h1}
\getchunk{include/hterror.h1}
\getchunk{include/lex.h1}
```

———-

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. */
```

———-

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))
```

———-

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);
```

———-

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;
```

———-

9.3 The Shared Code

— **htadd** —

```
\getchunk{htadd shared code}
```

———-

9.4 Code

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
}
```

———-

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);
}
```

———-

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;
    }
}
```

———-

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;
}
```

———-

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);
}
```

———-

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 10

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)

10.1 hthits Call Graph

This was generated by the GNU cflow program with the argument list. Note that the line:NNNN numbers refer to the line in the code after it has been tangled from this file.

```
cflow --emacs -l -n -b -T --omit-arguments hthits.c
```

```
;; This file is generated by GNU cflow 1.3. -*- cflow -*-
    2 {   0} +-main() <int main () line:279>
```

```
 3 {   1}   +-cmdline() <void cmdline () line:28>
 4 {   2}   | +-fprintf()
 5 {   2}   | \-exit()
 6 {   1}   +-regcomp()
 7 {   1}   \-handleHtdb() <void handleHtdb () line:38>
 8 {   2}     +-fopen()
 9 {   2}     +-badDB() <void badDB () line:269>
10 {   3}     | +-fprintf()
11 {   3}     | \-exit()
12 {   2}     +-getc()
13 {   2}     +-ungetc()
14 {   2}     +-handleFile() <void handleFile () line:53>
15 {   3}     | +-fgets()
16 {   3}     | +-sscanf()
17 {   3}     | +-stat()
18 {   3}     | +-fprintf()
19 {   3}     | +-exit()
20 {   3}     | +-ftell()
21 {   3}     | +-strncmp()
22 {   3}     | +-free()
23 {   3}     | +-malloc()
24 {   3}     | +-fseek()
25 {   3}     | +-strcmp()
26 {   3}     | +-strncpy()
27 {   3}     | +-badDB() <void badDB () line:269> [see 9]
28 {   3}     | \-handleFilePages() <void handleFilePages () line:138>
29 {   4}     |   +-fopen()
30 {   4}     |   +-fprintf()
31 {   4}     |   +-exit()
32 {   4}     |   +-handlePage() <void handlePage () line:151>
33 {   5}     |   | +-free()
34 {   5}     |   | +-malloc()
35 {   5}     |   | +-fprintf()
36 {   5}     |   | +-exit()
37 {   5}     |   | +-fseek()
38 {   5}     |   | +-fread()
39 {   5}     |   | +-splitpage() <void splitpage () line:211>
40 {   6}     |   | | +-fprintf()
41 {   6}     |   | | \-exit()
42 {   5}     |   | +-untexbuf() <void untexbuf () line:240>
43 {   6}     |   | | \-isalpha()
44 {   5}     |   | +-printf()
45 {   5}     |   | \-searchPage() <void searchPage () line:179>
46 {   6}     |   |   +-regexec()
47 {   6}     |   |   +-printf()
48 {   6}     |   |   +-squirt() <void squirt () line:197>
49 {   7}     |   |   | \-printf()
50 {   6}     |   |   \-strlen()
51 {   4}     |   \-fclose()
52 {   2}     \-fclose()
```

10.2 Constants and Headers

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>
```

———-

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
```

———-

structs

— **hthits** —

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

———-

Local includes

— **hthits** —

```
\getchunk{include/debug.h}
\getchunk{include/hthits.h1}
```

———-

local variables

— **hthits** —

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

———-

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];
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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");
    }
}
```

———-

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;
}
```

———-

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;
            }
    }
}
```

———-

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;
}
```

———-

badDB

— **hthits** —

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

———-

regerr

— **hthits** —

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

———-

main

— **hthits** —

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

———-

Chapter 11

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.

11.1 Constants and Headers

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>
```

———-

11.2 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;
```

———-

Local includes

— **hypertex** —

```
\getchunk{include/debug.h}
\getchunk{include/hyper.h}

\getchunk{include/all-hyper-proto.h1}
\getchunk{include/bsdsignal.h}
\getchunk{include/bsdsignal.h1}
\getchunk{include/hterror.h1}
\getchunk{include/pixmap.h1}
\getchunk{include/sockio-c.h1}
\getchunk{include/spadcolors.h}
\getchunk{include/spadcolors.h1}
\getchunk{include/util.h1}
```

———-

11.3 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;
```

———-

11.4 defines

— **hypertex** —

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

#define BACKCOLOR gControlBackgroundColor
#define below(y) ((y) + gWindow->page->scroll_off >= gWindow->page->bot_scroll_margin)
#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')
```

———-

11.5 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 */
```

———-

11.6 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;
```

———-

11.7 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;

\getchunk{hypertex shared code}
\getchunk{hashCopyEntry}
\getchunk{hashCopyTable}
\getchunk{dbFileOpen}
```

```
\getchunk{htperror}
\getchunk{dumpToken}
```

———-

11.8 Code

sigusr2Handler

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

— **hypertex** —

```
void sigusr2Handler(int sig) {
  gIsEndOfOutput = 1;
  return ;
}
```

———-

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);
}
```

———-

cleanSocket

Clean up spad sockets on exit.

— **hypertex** —

```
void cleanSocket(void) {
    char name[256];
    make_server_name(name, MenuServerName);
    unlink(name);
}
```

———-

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);
}
```

———-

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;
}
```

———-

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;
      }
  }
}
```

———-

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;
                }
            }
        }
    }
}
```

———-

11.9 Condition Handling

insertCond

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

```
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);
}
```

———-

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);
    }
}
```

———-

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;
}
```

———-

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;
    }
}
```

———-

11.10 Dialog Handling

redrawWin

— **hypertex** —

```
static void redrawWin(void) {
    XUnmapSubwindows(gXDisplay, gWindow->fMainWindow);
    XUnmapSubwindows(gXDisplay, gWindow->fScrollWindow);
    XFlush(gXDisplay);
    showPage(gWindow->page);
}
```

———-

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;
}
```

———-

incLineNumbers

— **hypertex** —

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

———-

decLineNumbers

— **hypertex** —

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

———-

decreaseLineNumbers

— **hypertex** —

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

———-

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();
}

/*
 */
```

———-

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;
        }
    }
}
```

———-

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);
}
```

———-

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);
}
```

———-

addBufferToSym

— **hypertex** —

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

```
        overwriteBuffer(buffer, sym);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
}
```

———-

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);
}
```

———-

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;
}
```

———-

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);
}
```

———-

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);
}
```

———-

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;
}
```

———-

backOverChar

— **hypertex** —

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

———-

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);
}
```

———-

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;
}
```

———-

deleteChar

— **hypertex** —

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

———-

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;
}
```

———-

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();
}
```

———-

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;
        }
}
```

———-

11.11 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.

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

11.12 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.

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);
      }
  }
}
```

———-

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;
    }

}
```

———-

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);
}

/*
 */
```

———-

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();
}
```

———-

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;
```

```
}
```

———-

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;
}
```

———-

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;
    }
}
```

———-

killAxiomPage

— **hypertex** —

```
static void killAxiomPage(HyperDocPage * page) {
    char command[512];
    sprintf(command, "(|htpDestroyPage| '%s)", page->name);
```

```
    sendLispCommand(command);
}
```

———-

killPage

— **hypertex** —

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

———-

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 */
```

———-

uplink

— **hypertex** —

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

———-

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;*/
}
```

———-

makeWindowLink

— **hypertex** —

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

———-

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;*/
    }
}
```

———-

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;
}
```

———-

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();
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
    }
}
```

———-

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);
}
```

———-

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
 */
```

———-

clearExposures

— **hypertex** —

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

———-

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;
  }
}
```

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);
}
```

———-

changeCursor

— **hypertex** —

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

———-

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);
}
```

———-

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);
    }
}
```

———-

initCursorStates

— **hypertex** —

```
static void initCursorStates(void) {
    hashMap(&gSessionHashTable,(MappableFunction) initCursorState);
}
```

———-

makeBusyCursor

— **hypertex** —

```
static void makeBusyCursor(HDWindow *window) {
    changeCursor(gBusyCursor, window);
}
```

———-

makeBusyCursors

— **hypertex** —

```
static void makeBusyCursors(void) {
    hashMap(&gSessionHashTable, (MappableFunction)makeBusyCursor);
}
```

———-

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);
}
```

———-

setErrorHandlers

— **hypertex** —

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

———-

11.13 Line Extent Computation

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;
}
```

———-

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;
}
```

———-

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;
}
```

————-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
        }
    }
}
```

———-

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;
}
```

———-

computeItemExtent

— **hypertex** —

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

———-

computeMitemExtent

— **hypertex** —

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

———-

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;
}
```

———-

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();
}
```

———-

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);
    }
}
```

———-

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();
}
```

———-

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();
}
```

———-

computeItExtent

— **hypertex** —

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

```
    node->y = text_y;
}
```

———-

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();
}
```

———-

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;
```

```
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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);
}
```

———-

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;
    }
}
```

———-

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;
    }
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
```

```
}

/*
 */
```

———-

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;
}
```

———-

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();
}
```

———-

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();
}
```

———-

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;
}
```

———-

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);
}
```

———-

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;
}
```

———-

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;
}
```

———-

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

```
        }
    }
    return 0;
}
```

———-

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;
}
```

———-

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;
    }
}
```

———-

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;
    }
}
```

———-

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;
}
```

———-

11.14 Handling forms

A few routines used to help with form extents

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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;
}
```

———-

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;
    }
}
```

———-

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;
    }
}
```

———-

11.15 Managing the HyperDoc group stack

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

emTopGroup

— **hypertex** —

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

———-

rmTopGroup

— **hypertex** —

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

———-

lineTopGroup

— **hypertex** —

```
void lineTopGroup(void) {
    if (! gTopOfGroupStack->next)
        pushGroupStack();
    gTopOfGroupStack->cur_color = gBorderColor;
    gTopOfGroupStack->cur_font = gRmFont;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);

}
```

———-

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);
}
```

———-

ttTopGroup

— **hypertex** —

```
void ttTopGroup(void) {
    if (! gTopOfGroupStack->next)
        pushGroupStack();
    gTopOfGroupStack->cur_color = gTtColor;
    gTopOfGroupStack->cur_font = gTtFont;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

———-

pushActiveGroup

— **hypertex** —

```
void pushActiveGroup(void) {
    pushGroupStack();
    gTopOfGroupStack->cur_font = gActiveFont;
    gTopOfGroupStack->cur_color = gActiveColor;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

———-

pushSpadGroup

— **hypertex** —

```
void pushSpadGroup(void) {
    pushGroupStack();
    gTopOfGroupStack->cur_font = gAxiomFont;
    gTopOfGroupStack->cur_color = gAxiomColor;
    changeText(gTopOfGroupStack->cur_color, gTopOfGroupStack->cur_font);
}
```

———-

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);
}
```

———-

centerTopGroup

— **hypertex** —

```
void centerTopGroup(void) {
    pushGroupStack();
    gTopOfGroupStack->center = 1;
}
```

———-

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;
}
```

———-

freeGroupStack

— **hypertex** —

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

———-

11.16 Handle input, output, and Axiom communication

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);
  }
}
```

———-

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);
  }
}
```

———-

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);
  }
}
```

———-

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;
}
```

———-

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;
}
```

———-

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);
        }
    }
}
```

———-

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|)");
          }
          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);
  }
 }
}
```

———-

strCopy

— **hypertex** —

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

———-

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;
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
    }
}
```

———-

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);
}
```

———-

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);
}
```

———-

11.17 X Window window initialization code

Initialize the X Window System.

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");*/
}
```

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;
}
```

———-

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);
}
```

———-

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;
}
```

———-

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);
}
```

———-

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;
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
        }
   }
}
```

———-

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");*/
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
}
```

———-

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);
}
```

———-

11.18 Handling user page interaction

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);
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
}
```

———-

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;
}
```

———-

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;
}
```

———-

11.19 Manipulate the item stack

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;
}
```

———-

clearItemStack

— **hypertex** —

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

———-

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);
}
```

———-

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;
}
```

———-

freeItemStack

— **hypertex** —

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

———-

11.20 Keyboard handling

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;
  }
}
```

———-

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;
}
```

———-

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);
    }
}
```

———-

11.21 Handle page macros

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;
}
```

———-

number

— **hypertex** —

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

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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);
    }
}
```

———-

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 ;
}
```

———-

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;
}
```

———-

11.22 Memory management routines

freeIfNonNULL

— **hypertex** —

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

———-

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;
}
```

———-

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);
  }
}
```

———-

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;
}
```

———-

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);
}
```

———-

allocIfnode

— **hypertex** —

```
IfNode *allocIfnode(void) {
  IfNode *tempif;
  tempif = (IfNode *) halloc(sizeof(struct if_node), "IfNode");
  tempif->thennode = tempif->elsenode = tempif->cond = NULL;
  return tempif;
}
```

———-

allocCondnode

— **hypertex** —

```
CondNode *allocCondnode(void) {
  CondNode *temp;
  temp = (CondNode *) halloc(sizeof(struct cond_node), "Cond Node");
  temp->cond = temp->label = NULL;
  return temp;
}
```

———-

freeCond

— **hypertex** —

```
static void freeCond(CondNode *cond) {
  if (cond) {
    free(cond->label);
    if (cond->cond)
      free(cond->cond);
    free(cond);
  }
}
```

———-

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;
}
```

———-

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;
  }
}
```

———-

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);
  }
}
```

———-

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);
}
```

———-

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);
}
```

———-

freeString

— **hypertex** —

```
void freeString(char *str) {
  freeIfNonNULL(str);
}
```

———-

freeDepend

— **hypertex** —

```
static void freeDepend(SpadcomDepend *sd) {
  freeIfNonNULL((char *) sd);
}
```

———-

dontFree

— **hypertex** —

```
static void dontFree(void  *link) {
  return;
}
```

———-

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);
  }
}
```

———-

freeInputItem

— **hypertex** —

```
void freeInputItem(InputItem *sym, short int des) {
  freeIfNonNULL(sym->name);
  freeLines(sym->lines);
  if (des)
    XDestroyWindow(gXDisplay, sym->win);
}
```

———-

freeInputList

— **hypertex** —

```
void freeInputList(InputItem *il) {
  while (il) {
    InputItem *trash = il;
    il = il->next;
    freeInputItem(trash, 0);
    free(trash);
  }
```

```
}
```

———-

freeInputBox

— **hypertex** —

```
static void freeInputBox(InputBox *box) {
  if (box) {
    freeIfNonNULL(box->name);
    free(box);
  }
}
```

———-

freeRadioBoxes

— **hypertex** —

```
static void freeRadioBoxes(RadioBoxes *radio) {
  if (radio) {
    freeRadioBoxes(radio->next);
    freeIfNonNULL(radio->name);
    free(radio);
  }
}
```

———-

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;
}
```

———-

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;
}
```

———-

allocPatchstore

— **hypertex** —

```
PatchStore *allocPatchstore(void) {
  PatchStore *p = (PatchStore *) halloc(sizeof(PatchStore), "PatchStore");
  p->loaded = 0;
  p->string = NULL;
  return p;
}
```

———-

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);
  }
}
```

———-

allocInputbox

— **hypertex** —

```
InputBox *allocInputbox(void) {
  InputBox *box = (InputBox *) halloc(sizeof(InputBox), "InputBox");
  box->picked = 0;
  box->next = NULL;
  box->rbs = NULL;
  return box;
}
```

———-

allocRbs

— **hypertex** —

```
RadioBoxes *allocRbs(void) {
  RadioBoxes *newrb = (RadioBoxes *) halloc(sizeof(RadioBoxes), "Radio Boxes");
  newrb->next = NULL;
  newrb->boxes = NULL;
  return newrb;
}
```

———-

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;
}
```

———-

freeButtonList

— **hypertex** —

```
void freeButtonList(ButtonList *bl) {
  while (bl) {
    ButtonList *nbl = bl->next;
    free(bl);
    bl = nbl;
  }
}
```

———-

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;
}
```

———-

11.23 Page parsing routines

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;
}
```

———-

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);
    }
}
```

———-

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;
    }
}
```

———-

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);
}
```

———-

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 */
```

———-

parseFromString

— **hypertex** —

```
void parseFromString(char *str) {
    saveScannerState();
    last_ch = NoChar;
    last_token = 0;
    inputString = str;
    inputType = FromString;
    parseHyperDoc();
    restoreScannerState();
}
```

———-

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();
}
```

———-

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
 */
```

———-

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;
}
```

———-

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;
}
```

———-

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);
}

/*
 */
```

———-

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;
        }
    }
}
```

————-

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;
}
```

———-

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;
}
```

———-

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;
}
```

————-

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;
}
```

———-

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;
    }
}
```

———-

parseReplacepage

— **hypertex** —

```
static void parseReplacepage(void) {
    getExpectedToken(Lbrace);
    getToken();
    replace_page = allocString(token.id);
    getExpectedToken(Rbrace);
}
```

———-

windowEqual

Hash functions for active link windows.

— **hypertex** —

```
int windowEqual(Window *w1, Window *w2) {
    return *w1 == *w2;
}
```

———-

windowCode

Hash code for a window.

— **hypertex** —

```
int windowCode(Window *w, int size) {
    return (*w) % size;
}
```

———-

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);
}
```

———-

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);
}
```

———-

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); */
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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");
}
```

———-

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);
        }
    }
}
```

———-

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;
}
```

———-

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);
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

11.24 Handle InputString, SimpleBox, RadioBox input

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 */
```

———-

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;
}
```

———-

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;
}
```

———-

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 ;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
  }
}
```

———-

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;
  }
}
```

———-

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;
}
```

———-

currentItem

— **hypertex** —

```
InputItem *currentItem(void) {
  InputItem *trace = gPageBeingParsed->input_list;
  if (trace) {
    for (; trace->next != NULL; trace = trace->next);
    return trace;
  }
  else
    return NULL;
}
```

———-

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;
}
```

———-

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;
}
```

———-

11.25 Routines for paste-in areas

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--;
}
```

———-

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);
}
```

———-

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;
}
```

———-

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();
}
```

———-

11.26 parsing routines for node types

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--;
}
```

———-

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;
    }
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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++;
}
```

———-

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();
    }
}
```

———-

parseMitem

— **hypertex** —

```
void parseMitem(void) {
    if (!gInItems) {
        fprintf(stderr, "\\mitem found outside an items environment\n");
        printPageAndFilename();
        printNextTenTokens();
        jump();
    }
    curr_node->type = Mitem;
}
```

———-

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;
}
```

———-

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);
        }
    }
}
```

———-

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;
}
```

———-

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--;
}
```

———-

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--;
}
```

———-

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--;
}
```

———-

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--;
}
```

———-

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);
}
```

---

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;
}
```

———-

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;
}
```

———-

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;
    }
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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();
    }
}
```

———-

11.27 Reading bitmaps

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

11.28 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

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);
}
```

———-

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);
}
```

———-

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

  ************************************************************************/
```

———-

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);
}
```

———-

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();
    }
}
```

———-

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));
}
```

———-

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);
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
    }
}
```

———-

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);
}
```

———-

hideScrollBars

— **hypertex** —

```
void hideScrollBars(HDWindow * hdWindow) {
    XUnmapWindow(gXDisplay, hdWindow->fScrollDownWindow);
    XUnmapWindow(gXDisplay, hdWindow->fScrollUpWindow);
    XUnmapWindow(gXDisplay, hdWindow->scrollbar);
    XUnmapWindow(gXDisplay, hdWindow->scroller);
}
```

———-

getScrollBarMinimumSize

— **hypertex** —

```
void getScrollBarMinimumSize(int *width, int *height) {
    (*width)  = sup_width + 4;
    (*height) = sup_height + sdown_height + 5;
}
```

———-

ch

— **hypertex** —

```
static int ch(int height) {
    int rem = height % line_height;
    if (rem == 0)
        return height;
    return height - rem + line_height;
}
```

———-

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);
    }
}
```

———-

11.29 Display text object

/* * Display the page whose extent has been computed, using the actual size of * the window, and yOff to determine clipped areas */

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;
        }
    }
}
```

———-

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();
}
```

———-

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();
}
```

———-

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);
}
```

———-

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);
}
```

———-

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);
    }
}
```

———-

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);
}
```

———-

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;
    }
}
```

———-

11.30 Axiom communication interface

Still a problem with closeClient.

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;
}
```

———-

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);
}
```

———-

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);
                    }
                }
}
```

———-

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;
                }
}
```

————-

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);
}
```

———-

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;
        }
}
```

———-

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;
```

```
}
```

———-

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;
                }
            }
        }
    }
}
```

———-

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);
}
```

———-

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
 */
```

———-

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;
}
```

———-

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;
}
```

———-

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;
}
```

———-

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;
```

```
}
```

———-

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;
}
```

———-

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;
    }
}
```

———-

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);
}
```

———-

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);
}
```

———-

escapeString

— **hypertex** —

```
void escapeString(char *s) {
    char *st;
```

```
    for (st = s; *st; st++)
        *st = funnyEscape(*st);
}
```

———-

unescapeString

— **hypertex** —

```
void unescapeString(char *s) {
    char *st;
    for (st = s; *st; st++)
        *st = funnyUnescape(*st);
}
```

———-

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;
          }
    }
}
```

———-

printSourceToString

— **hypertex** —

```
char *printSourceToString(TextNode *command) {
    int len = 0;
    printSourceToString1(command, &len);
    p2sBuf = resizeBuffer(len, p2sBuf, &p2sBufSize);
    return printSourceToString1(command, NULL);
}
```

———-

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;
}
```

———-

11.31 Produce titlebar

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);
}
```

———-

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();
}
```

———-

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);
}
```

———-

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);
}
```

———-

getTitleBarMinimumSize

— **hypertex** —

```
void getTitleBarMinimumSize(int *width, int *height) {
    (*width)  = 4 * twwidth + 40;
    (*height) = twheight + 2;
}
```

———-

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);
        bsdSignal(SIGCHLD, sigcldHandler,RestartSystemCalls);
        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 12

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/presea case=0 expr=
fi
```

————-

Chapter 13

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";
}
```

———-

13.1 token.h

— **include/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 14

The Bitmaps

14.1 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};
```

———-

14.2 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};
```

———-

14.3 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};
```

———-

14.4 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};
```

———-

14.5 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};
```

———-

14.6 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};
```

————

14.7 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};
```

14.8 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};
```

14.9 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};
```

14.10 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};
```

———-

14.11 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};
```

———-

Bibliography

[1] nothing